RasPi

DESIGN
BUILD
CODE

25

Get hands-on with your Raspberry Pi

VISUALISE MUSIC WITH LEDs

SEND TEXTS WITH PI

Plus

Stream media through your network

Welcome

The Raspberry Pi is the perfect size for a lot of things, but one of the most fun things that you can do with it is have it control LEDs. Who wouldn't want their own Pi-based disco? And that's exactly what you'll learn to create this issue, as we demonstrate how you can get the Pi to make a five-metre strip of LEDs react to music. Want to see a similar idea on an even grander scale? Check out ElectroSuper, in which Fred Sapey-Triomphe and Yann Guidon apply the same principle to an entire railway station. It turns out that despite its diminutive size, you can still think big with the Pi. Also, learn to use your Pi as a networked media player, control sensors and send text messages. Enjoy the issue!

April

Editor

Get inspired

Discover the RasPi community's best projects

Expert advice

Got a question? Get in touch and we'll give you a hand

Easy-to-follow guides

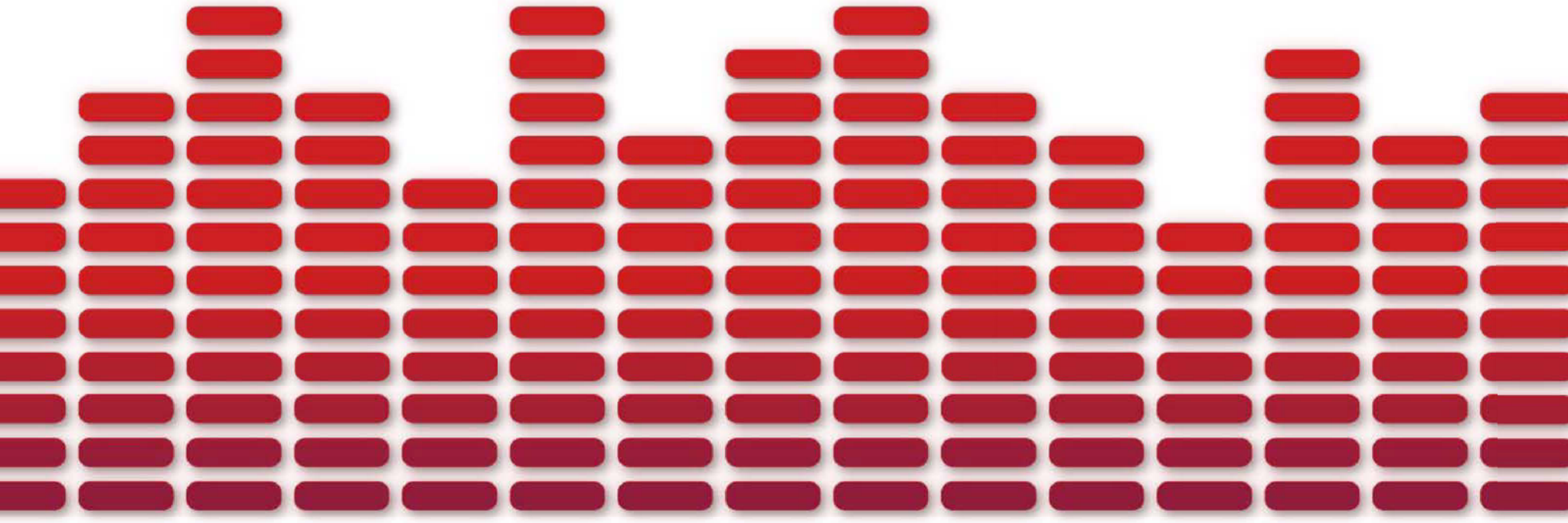
Learn to make and code gadgets with Raspberry Pi

From the makers of

LinuxUser & Developer

Join the conversation at...

@linuxusermag | Linux User & Developer | RasPi@imagine-publishing.co.uk



Contents

Visualise music with LEDs

Get the party started with a five-metre LED strip that reacts to music

In this feature, we will write a Python script that takes audio as the input, either as a line-in stream from an audio card or as a WAVE file to play. This audio stream will then be filtered into three bands – bass, middle and treble – using the signal-processing library that comes with SciPy. The amplitude of each band is averaged and then mapped to a colour on an LED strip in real time, so that the LED strip changes colour as the energy of the music changes. The amazing results are sure to be the talk of your party.

The script uses roughly 20% of one CPU core on a Raspberry Pi 2. As a result, it is highly likely that this tutorial will work just fine on any older Raspberry Pi model that may be collecting dust in your drawer. This is an excellent opportunity to get that older model out, blow off the dust and put it to good use.

THE PROJECT ESSENTIALS

LED strip

USB sound card (we used a Behringer UCA202)

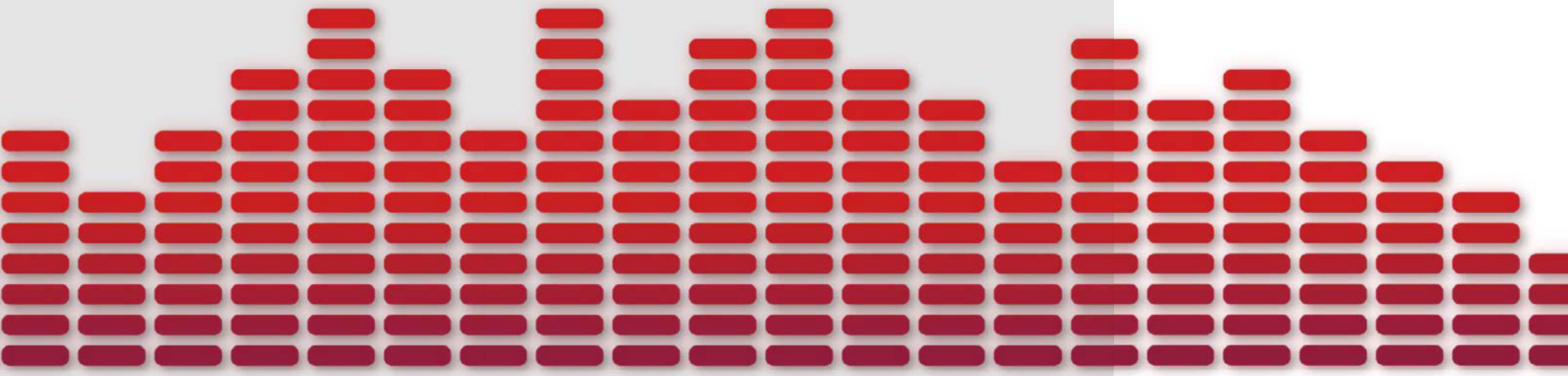
Breadboard

Female to male jumper cables

3 x TIP120 NPN transistors

3 x 220Ω resistors

DC jack



01 Install dependencies

Start with a fresh Raspbian image and upgrade it to the latest package listings with:

```
sudo apt-get update; sudo apt-get upgrade
```

Then install the required dependencies and next compile PyAudio. PortAudio is used by PyAudio. Pip is used to compile PyAudio as it is not included with Raspbian by default. SciPy is used to filter the audio, while matplotlib is used to plot the frequency response of the filters.

```
sudo apt-get install python-pip python2.7-
    dev portaudio19-dev python-scipy python-
    matplotlib
sudo pip install pyaudio
```

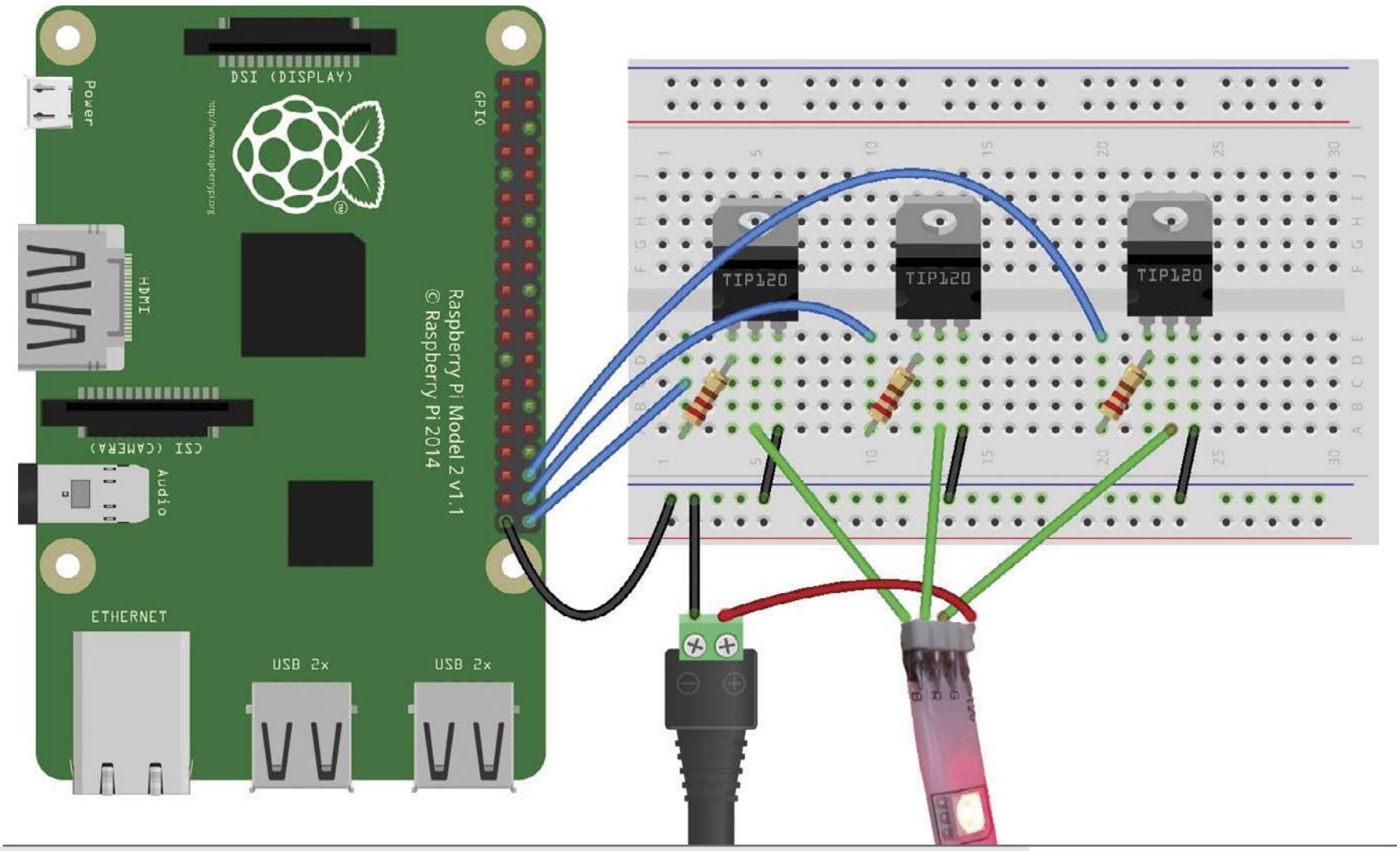
03 Test the sound card output

If you're using the Pi to play audio rather than a line-in, then you'll want to test the output. Type alsamixer and then make sure the volume is set to a comfortable level. If you are plugging speakers in, then set to 100%. Then type speaker-test, this will generate pink noise on the speakers. Press Ctrl+C to exit if you're happy it's working. Sadly, there's no easy way to test a line-in signal. It will be obvious if it's working once the software is written.

04 Construct the circuit

We are using GPIO pins 20, 21 and 16 for red, green and blue, respectively. These pins go through a 220Ω resistor to the first pin of the transistor (base). The

"This project also works with line-in, which the Pi doesn't have, so a USB sound card is ideal"



second pin (collector) goes to one of the red, green or blue LED-strip pins. The third pin (emitter) goes to ground. The ground from the DC jack needs to be connected to the same ground as the Pi. Connect the 12V from the DC jack to the strip. When connecting the wires, ensure the wires aren't touching. If anything shorts, that would be bad. Ideally, this circuit would go on a stripboard and be soldered to the LED strip.

Above Diagram of how to connect the wires to your pins – see Step 4. Remember not to let the wires touch

05 Start the script

Create a file called ledstrip.py and mark it as executable. Then begin editing with your favourite editor:

```
touch ledstrip.py
chmod +x ledstrip.py
vim ledstrip.py
```

06 Add imports

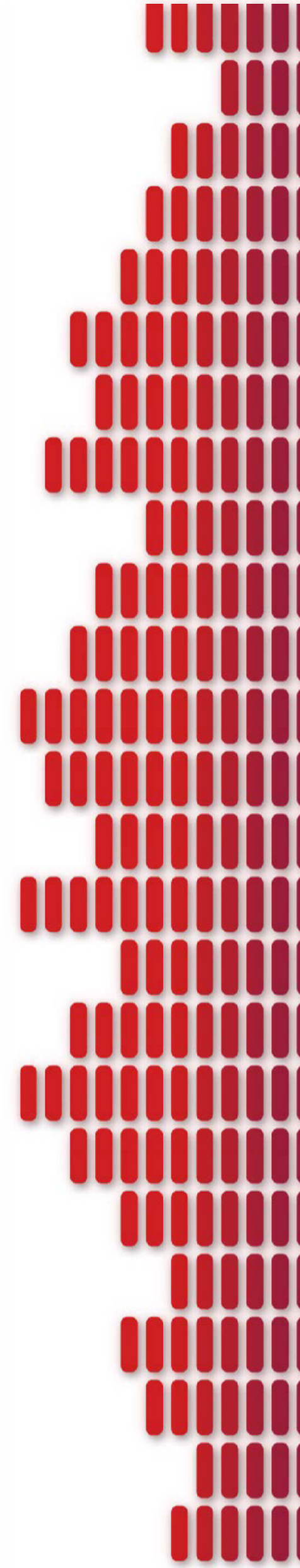
After adding the shebang (#!) line, you'll need to add the imports. The GPIO library is used for sending signals to the LED strip. Randrange is used to generate random colours. PyAudio is the audio library. Wave is used to read WAVE files. From SciPy, butter is the filter type that we're using, lfilter is used to apply the filter and freqz gets the frequency response. Matplotlib is used to plot the frequency response of the filters and NumPy is used for fast-math operations on arrays. The back-end of NumPy is written in C, which means it's faster than Python when doing the same operation on a large data set.

07 Create LED class

The LED controller class is simple. The init function will take a tuple containing (R, G, B) pin numbers and set up the pins. There is a set_colour function that takes an (R, G, B) tuple where the values are between 0 and 255. Finally, there is a test function that sets a random colour every second. This class could be reused in a different project with no modifications. The class is well commented so doesn't need any further explanation.

08 Create the frequency analyser

The frequency analyser class is responsible for taking the audio data, then filtering it into either bass, middle or treble, and then controlling the LEDs. As arguments, it takes the number of channels in the audio data (assumed to be one or two), the sample rate (usually 44100Hz) and an instance of the LED-controller class.



09 The init method

After storing the input parameters, we calculate the Nyquist frequency, which is half of the sample rate. Nyquist's theorem says that the sample rate of any analogue signal needs to be at least twice the analogue frequency. This is why CDs are sampled at 44.1KHz (human hearing ends at around 20KHz).

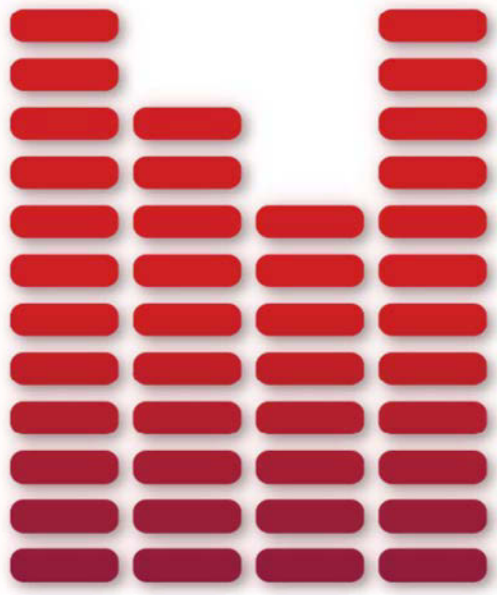
Then design two butterworth filters (a filter designed to have a frequency response as flat as possible). The low-pass filter cut-off is 200Hz and the high-pass cut-off is 4000Hz. The parameters are later used to filter with these characteristics.

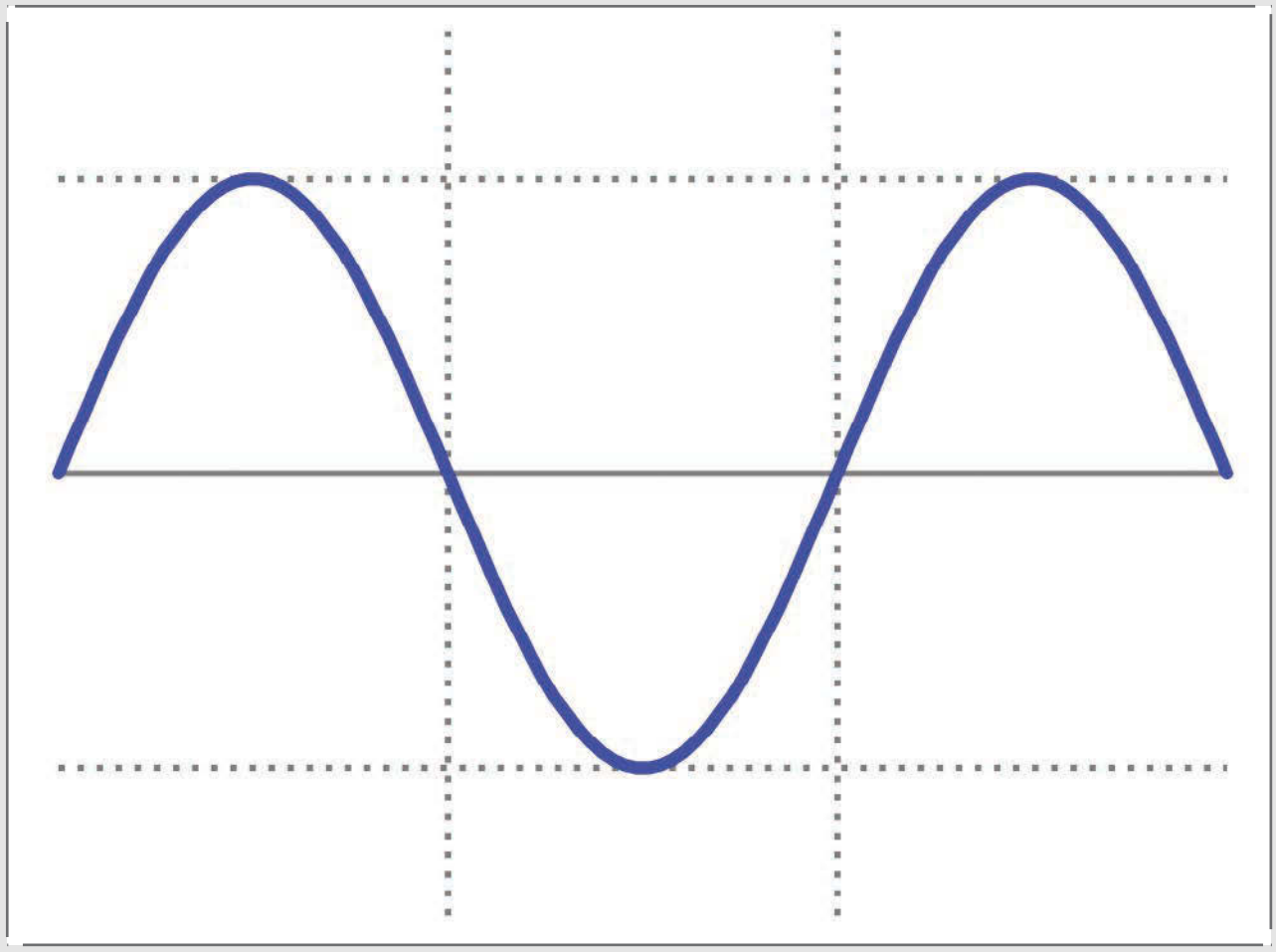
10 Colours

Finally, we want the colour changes to the strip to be pronounced and noticeable, with reduced flickering. To achieve this, we can store the max value and use it to have colours fall by a certain amount until the energy of the song pushes them back up. The fall list sets the rate for each colour/frequency in the order (R, G, B/ bass, mids, treble). The range of values for a colour is between 0 and 255. The LEDs are set at a frequency of Sample Rate/CHUNK. CHUNK is the amount of data to read at once. We set CHUNK to 512, so 44100/512 = 86 times a second. Keep this in mind when setting your fall values, as they depend on what you think looks good and the style of music.

11 The filter function

The filter function is fairly straightforward. An array of samples called data is passed in where the samples are values between -1.0 and 1.0. This is the standard





Left A perfect sine wave – see Step 12

format for signal processing, as opposed to the 16-bit integers used in WAVE files and CDs. The low- and high-pass filters are then applied using the parameters we calculated before in Step 9. We then get the middle band by subtracting the sum of the low and high filter from the original signal.

12 Root mean square

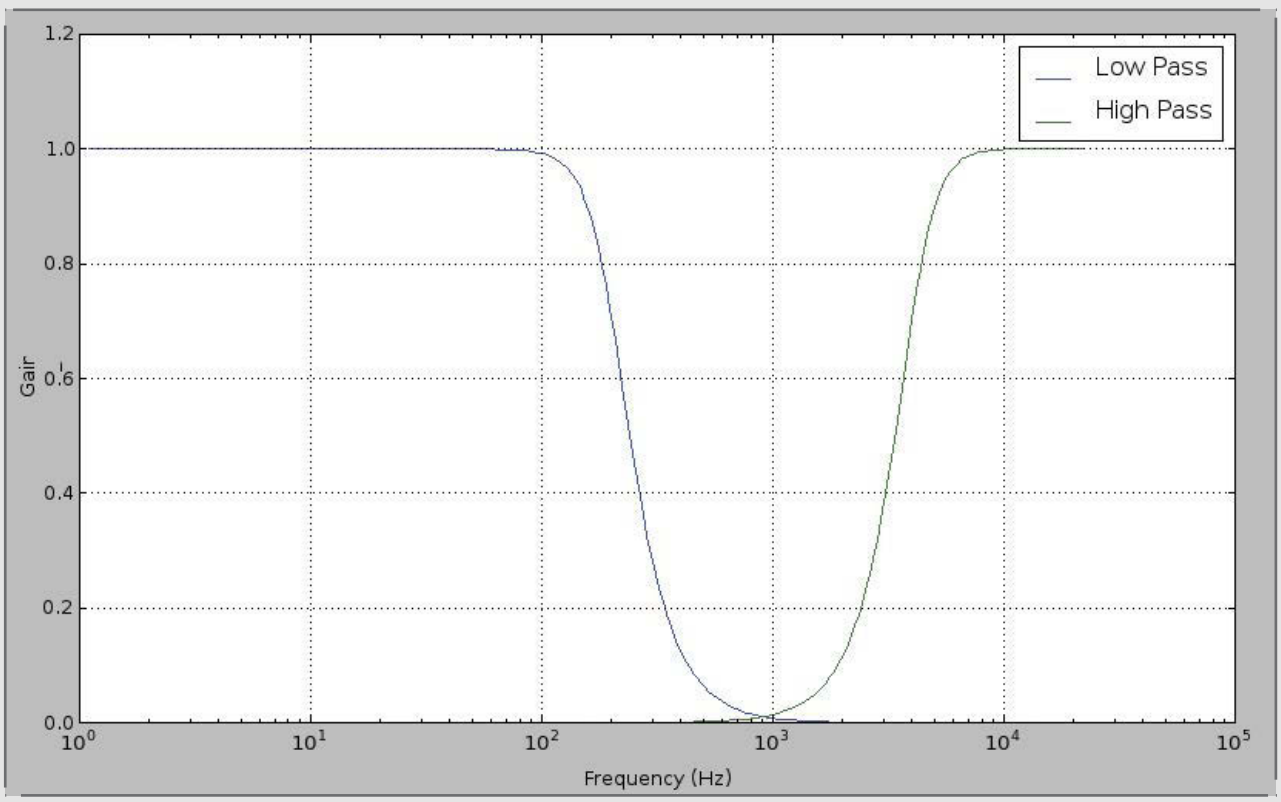
The root mean square, or quadratic mean, is used to get the average amplitude of the samples in each frequency band. Why are the values squared and rooted? Imagine a perfect sine wave (as shown in the image) that goes between -1 and 1. If you averaged the samples of one cycle using the normal mean method, then the value would be zero because the positive and negative samples would cancel each other out.

13 Change LEDs

Now comes the fun part. The change_leds function uses the filtered data, gets the average amplitude of each set and converts it back to a value between 0 and 255. After that, you have the opportunity to fiddle with the values to emphasise frequency values. It is these and the values of the fall array from Step 10 that will determine how your LED strip behaves. When setting a colour, we update the max if the new value is higher than the previous one. If not, we decrement the max by the fall value and use that as the colour to set. This makes the colours fall in a controlled manner when the energy drops. Also, make sure you don't go out of bounds when doing these calculations.

14 Plot frequency response

The plot_response function isn't a necessary part of the program, but we used it to draw the frequency response of the filters to ensure they would behave as expected. As such, it is nice to have if you want to have a go at changing the filters. The frequency scale is a log scale because that's how human hearing works. If you log into



the Pi with ssh -X pi@ip.address and call the function, you should get the plot forwarded to your local machine.

15 The audio controller

The audio controller is the last piece of the puzzle. It is responsible for either playing a WAVE file or capturing a line-in signal, and then sends that data off to be filtered and displayed on the LED strip. The init method takes a filename (or line-in) and an instance of the LED Controller class. A flag is set indicating if the line input is being used. If not, the WAVE file is opened. Finally, the LED instance is stored and an instance of the PyAudio library is created.

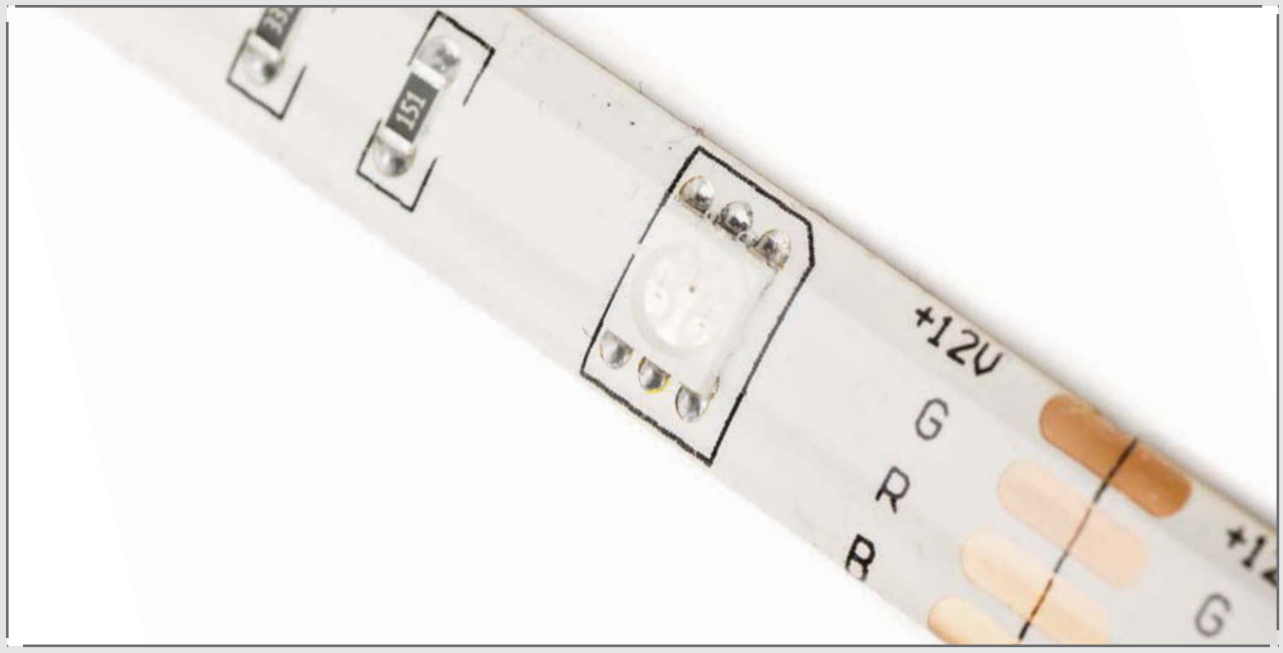
16 Help methods

There are a couple of utility methods in the audio controller. The first one gets the left-hand side of a stereo signal because there's no point analysing both sides. There is also a more function, which gets another chunk of audio data from either the WAVE file or line input. Oddly, there is always a line-in error when reading from it the first time. It should stay stable after this – if not, try changing the chunk size. If there is an error, we just return random data.

17 Analyse method

The analyse method is responsible for converting the byte string returned by the more method into an array of 16-bit integers. Then if the audio is stereo, the right-hand side is discarded. The data is then converted into floating-point representation where each sample is between -1.0 and 1.0. This is the standard data format

"The analyse method converts the byte string returned by the more method into an array of 16-bit integers"



Left The strip we're using can be bought here: **http://amzn.to/1Lkutx6**

for signal processing. The floating-point data is then sent to the frequency analyser. Finally, the change_leds method of the frequency analyser is called.

18 Setup methods

The main function of the audio controller is the loop method. Before the loop begins, a setup helper function is called, which initialises PyAudio to either record from line-in or play back a WAVE file.

19 Loop method

As previously mentioned, the main processing loop begins by setting up the appropriate audio stream. It then initialises the frequency analyser and starts a while loop that runs until there is no audio data left. The audio data is sent to the sound card if we are playing a WAVE file and then sent to the analyser, which deals with the actual analysis and changes the LEDs.

20 The main method

The main method is really simple to perform. The first two lines are for plotting the frequency response of the filters. After this, we create an instance of the LED Controller by passing a tuple containing the GPIO

pin numbers for the (red, green, blue) transistors. We then create an instance of the Audio Controller, which passes through both the first command line argument (either line-in or a WAVE file) and also the LED Controller instance we just created. Finally, we enter the processing loop.

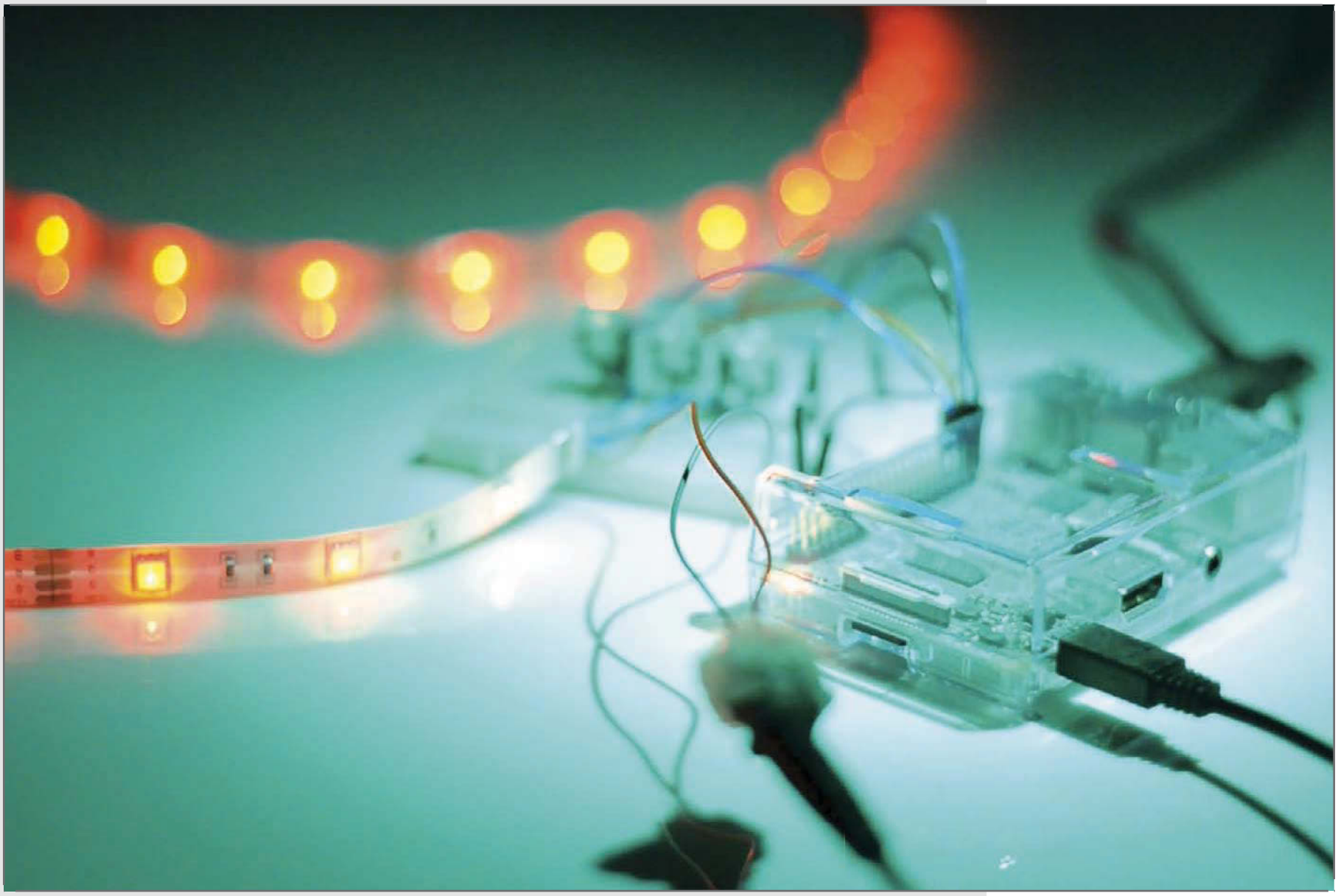
21 Start at boot

Now that we have the code finished, it's time to make it start at boot. The application needs to run as root, so we can simply add it to rc.local. Edit /etc/rc.local with sudo editor and add the line:

```
python2 /home/pi/ledstrip.py line-in &
```

… before the exit 0 line. Now reboot and test your visualiser!

Below Keep the LED strip coiled as a single unit or unroll and drape it around your subwoofers – the choice is yours!



The Code VISUALISE WITH LEDS

05

```
#!/usr/bin/env python2

import RPi.GPIO as GPIO
from random import randrange
import time
import pyaudio
import wave
from scipy.signal import butter, lfilter, freqz
import matplotlib
#matplotlib.use("GTK") #uncomment when plotting
import matplotlib.pyplot as plt
import numpy as np
import sys
```

07

```
# How many bytes of audio to read at a time
CHUNK = 512


class LedController:
    def __init__(self, pin_nums):
        # pin_nums is an R, G, B tuple

        # Initial setup of GPIO pins
        GPIO.setmode(GPIO.BCM)

        # Set each pin as an output and create a pwm
instance
        self.pins = []
        for p in pin_nums:
            GPIO.setup(p, GPIO.OUT)
            # Create a pwm instance for the pin at a
            # frequency of 200Hz
            self.pins.append(GPIO.PWM(p, 200))
            # Set each pin to a random brightness to
begin with
```

The Code VISUALISE WITH LEDS

07

```
            self.pins[-1].start(randrange(0, 100))

    def set_colour(self, colour_tuple):
        # Takes a colour tuple in the form (R, G, B)
where the
        # values are from 0 to 255 > 255 is capped

        for i in range(0, 3):
            # Scale 0 to 255 to a percentage
            scaled = int(colour_tuple[i] *
                         (100.0/255.0))

            # Ensure we are giving correct values
            if scaled < 0:
                scaled = 0.0
            elif scaled > 100:
                scaled = 100.0

            #print "{0}: {1}".format(i ,scaled)
            self.pins[i].ChangeDutyCycle(scaled)

    def test(self):
        # Change to a random colour
        while True:
            r = randrange(0, 256)
            g = randrange(0, 256)
            b = randrange(0, 256)
            self.set_colour((r, g, b))
            time.sleep(1)
```

09

```
class FreqAnalyser:
    # Filtering based on
    # http://wiki.scipy.org/Cookbook/ButterworthBandpass
```

The Code VISUALISE WITH LEDS

09

```
    def __init__(self, channels, sample_rate,
leds=None):
        self.leds = leds # Not needed if just plotting
        self.channels = channels
        self.sample_rate = sample_rate
        self.nyquist = float(sample_rate) / 2

        # Filter order - higher the order the sharper
        # the curve
        order = 3

        # Cut off frequencies:
        # Low pass filter
        cutoff = 200 / self.nyquist
        # Numerator (b) and denominator (a)
        # polynomials of the filter.
        b, a = butter(order, cutoff, btype='lowpass')
        self.low_b = b
        self.low_a = a

        # High pass filter
        cutoff = 4000 / self.nyquist
        b, a = butter(order, cutoff, btype='highpass')
        self.high_b = b
        self.high_a = a
```

10

```
        # Keep track of max brightness for each colour
        self.max = [0.0, 0.0, 0.0]
        # Make different frequencies fall faster
        # bass needs to be punchy.
        self.fall = [15.0, 2.5, 5.0]
```

11

```
    def filter(self, data):
        # Apply low filter
```

The Code VISUALISE WITH LEDS

11
```
        self.low_data = lfilter(self.low_b,
                                self.low_a,
                                data)

        # Apply high filter
        self.high_data = lfilter(self.high_b,
                                 self.high_a,
                                 data)

        # Get mid data by doing signal - (low + high)
        self.mid_data = np.subtract(data,
                        np.add(self.low_data,
                               self.high_data))
```

12
```
    @staticmethod
    def rms(data):
        # Return root mean square of data set
        # (i.e. average amplitude)
        return np.sqrt(np.mean(np.square(data)))
```

13
```
    def change_leds(self):
        # Get average amplitude
        l = []
        l.append(self.rms(self.low_data))
        l.append(self.rms(self.mid_data))
        l.append(self.rms(self.high_data))

        # These values are floating point from 0 to 1
        # and our led values go to 255
        divval = 1.0/255

        for i in range(0, 3):
            l[i] = l[i] / divval
```

13

```
        # Do any number fudging to make it look better
        # here - probably want to avoid high values of
        # all because it will be white
        l[0] *= 2 # Emphasise bass
        l[1] /= 2 # Reduce mids
        l[2] *= 5 # Emphasise treble
        #print l

        for i in range(0, 3):
            # First cap all at 255
            if l[i] > 255.0:
                l[i] = 255.0

            # Use new val if > previous max
            if l[i] > self.max[i]:
                self.max[i] = l[i]
            else:
                # Otherwise, decrement max and use that
                # Gives colour falling effect
                self.max[i] -= self.fall[i]
                if self.max[i] < 0:
                    self.max[i] = 0
                l[i] = self.max[i]

        self.leds.set_colour(l)
```

14

```
    def plot_response(self):
        # Frequency response of low and high pass
        # filters. Borrowed from
        # http://wiki.scipy.org/Cookbook/
ButterworthBandpass
        plt.figure(1)
        plt.clf()
        w, h = freqz(self.low_b,
```

14

```
                    self.low_a,
                    worN=20000)
        plt.plot((self.nyquist / np.pi) * w,
                 abs(h), label="Low Pass")

        w, h = freqz(self.high_b,
                    self.high_a,
                    worN=20000)
        plt.plot((self.nyquist / np.pi) * w,
                 abs(h), label="High Pass")

        plt.xlabel('Frequency (Hz)')
        plt.ylabel('Gain')
        plt.grid(True)
        plt.legend(loc='best')

        plt.xscale('log')
        plt.show()
        # Exit at after showing the plot. Only to
        # verify frequency response
        sys.exit()
```

15

```
class AudioController:
    def __init__(self, filename, leds):
        if filename == 'line-in':
            self.line_in = True
        else:
            self.line_in = False
            self.wf = wave.open(filename)

        self.leds = leds
        self.p = pyaudio.PyAudio()
```

The Code VISUALISE WITH LEDS

16

```
    @staticmethod
    def get_left(data):
        # Return the left channel of stereo audio
        data = np.reshape(data, (CHUNK, 2))
        return data[:, 0]
    def more(self):
        if self.line_in:
            try:
                # Return line in data
                return self.stream.read(CHUNK)
            except:
                print "line-in error"
                return 'ab'
        else:
            # Read data from wav file
            return self.wf.readframes(CHUNK)
```

17

```
    def analyse(self, data):
        # Convert to numpy array and filter
        data = np.fromstring(data, dtype=np.int16)

        # If stereo only work on left side
        if self.channels == 2:
            data = self.get_left(data)

        # Convert int16 to float for dsp
        data = np.float32(data/32768.0)

        # Send to filter
        self.analyser.filter(data)

        self.analyser.change_leds()
```

18

```
    def play_setup(self):
        # Assume 16 bit wave file either mono or stereo
        self.channels = self.wf.getnchannels()
        self.sample_rate = self.wf.getframerate()
        self.stream = self.p.open(format = pyaudio.
paInt16,
                                  channels = self.
channels,
                                  rate = self.sample_
rate,
                                  output = True)

    def record_setup(self):
        self.channels = 1
        self.sample_rate = 44100
        self.stream = self.p.open(format = pyaudio.
paInt16,
                                  channels = self.
channels,
                                  rate = self.sample_
rate,
                                  input = True)
```

19

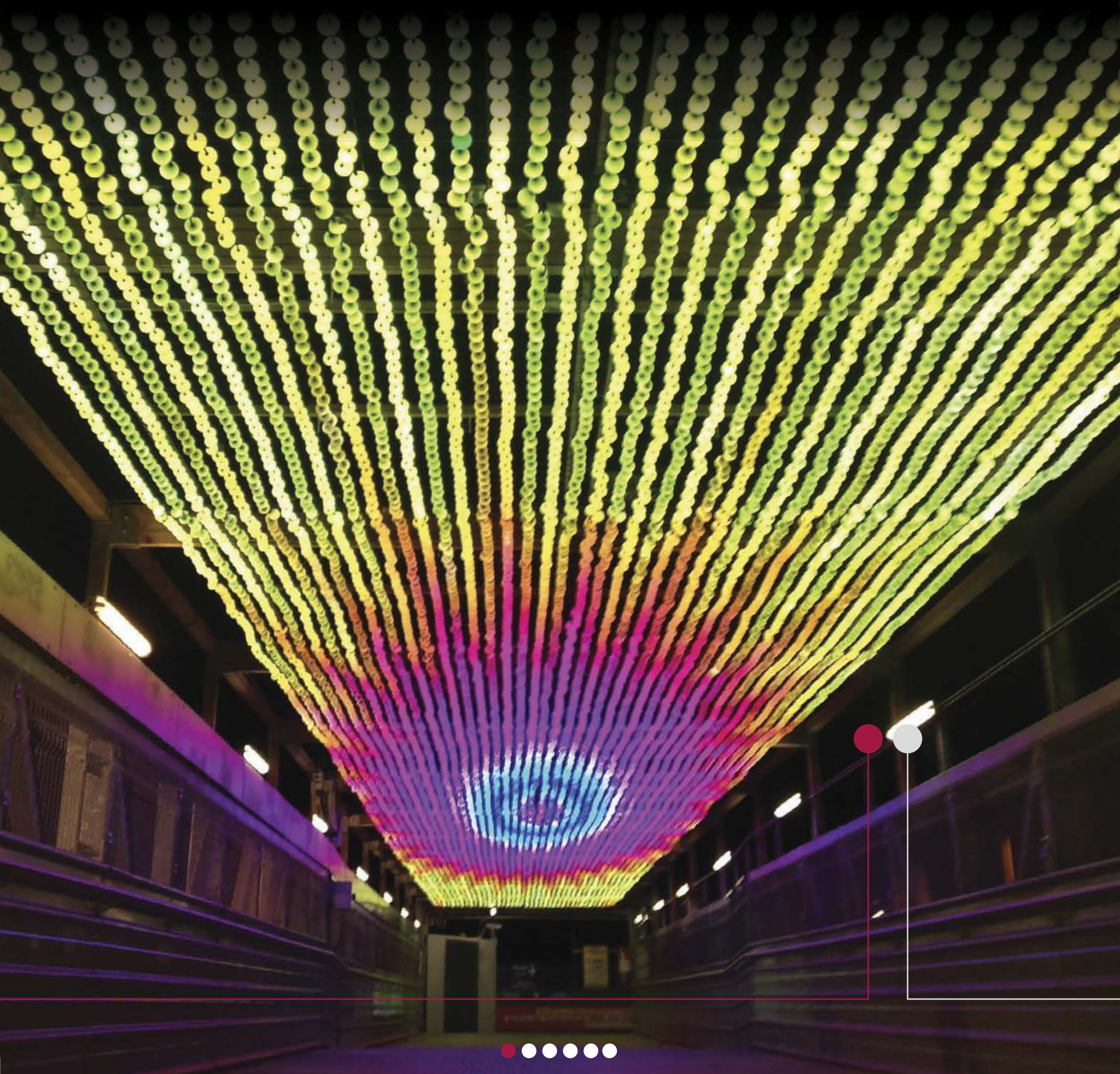
```
    def loop(self):
        # Main processing loop
        # Do appropriate setup depending on line in or
not
        if self.line_in:
            self.record_setup()
        else:
            self.play_setup()

        self.analyser = FreqAnalyser(self.channels,
```

19

```
                                    self.sample_rate,self.
leds)

        # Read the first block of audio data
        data = self.more()

        # While there is still audio left
        while data != '':
            try:
                # If we're playing audio write to
stream
                if not self.line_in:
                    self.stream.write(data)

                # Analyse data and change LEDs
                self.analyse(data)

                # Get more audio data
                data = self.more()
            except KeyboardInterrupt:
                break

        # Tidy up
        self.stream.close()
        self.p.terminate()

20
if __name__ == "__main__":
    #f = FreqAnalyser(2, 44100)
    #f.plot_response()

    lc = LedController((20, 21, 16))
    #lc.test()
    ac = AudioController(sys.argv[1], lc)
    ac.loop()
```

ElectroSuper

Fred Sapey-Triomphe and Yann Guidon make Mons railway station sparkle with a supersized LED installation



The ElectroSuper installation at Mons station looks amazing! How long have you been collaborating?

Fred Yann and I have been working together since February 2013. I wanted to create a large-scale LED display, and I'm a visual artist and have no foundation in electronics, so I couldn't make it by myself. I met Yann through common friends and I went to his studio early in 2013. I was asked to do another project for a show in Normandy, France, so I had a little budget for that and I asked Yann if he would like to start working on that project.

How did you begin work on the ElectroSuper project together?

Fred Well, basically we had a show in 2014 in Pompidou Centre, in the east of France, close to the German border in Metz and very close to Belgium. And there is a guy there in Metz who told me that the city of Mons is looking for something to dress up the railway station. [Ed: Mons was a European Capital of Culture in 2015.] This guy said, "Why don't you send your portfolio to Mons?" So that's what I did and we finally signed the contract three months before the opening. We only had two months to produce the whole piece, which is a 42 metre long ceiling screen.

The screen interacts with passers-by in the tunnel – how does that work?

Fred The idea was to cover the ceiling of a passenger path. People getting off the train have to take this path to the station, so this is the first thing visitors are going to see as they arrive. We were asked to create something engaging, powerful, colourful, something

Fred Sapey-Triomphe is a visual artist who uses light as raw material. Trained at the École nationale supérieure des Beaux-Arts and the École Boulle, he lives and works in Paris

Yann Guidon is an engineer who works with artists, specialising in electronics and algorithmics. He has designed two open source CPU architectures: F-CPU and YASEP

that would put the visitor in a good mood for their visit. We wanted it to be interactive, so Yann put in four infrared sensors, at the entry and exit points of the tunnel. The images that are displayed by the screen are changing according to the number of visitors.

Yann I put each pair of sensors one metre apart, so when I pick up a series of pulses I know that something is moving in one direction on the pathway.

So it's kind of tidal, then – every time there's a new movement, it sends another wave of colour?

Fred Yes, it's a good way to explain it. Also, this project is running for the whole year. We designed it so that the visual effect varies according to the seasons. So right now it's August and the amount of light is larger than in December, so we had to create specific videos for each season.

Aside from the sensors, what other hardware are you using?

Yann Fred designed the whole structure and helped build it. There are a lot of wooden structures with special treatment for the wood because it has to sustain snow, rain and sun. He found premade elements on which we could affix LED strips, 2 by 3.5 metres, like a tile. So we split the whole surface into six sections; each section is 40 by 70 bulbs, so 2,800 bulbs. We have 16,800 in total and it's about one watt per bulb, so if you multiply everything you get more than 16 kwatts. The length is 42 metres and we have to transmit data across this distance. It creates a problem of integrity, reliability, etc, so I chose to use Ethernet for

THE PROJECT ESSENTIALS

- Raspberry Pi B+
- Real-time clock module
- 8-port Ethernet hub
- Raspberry Pi power supply
- 6 x WizYasep boards
- 120 x 150W power supplies
- 50,400 x RGB LEDs (3/bulb)
- 4 x infrared sensors

the transmission of data because it's cheap and well supported.

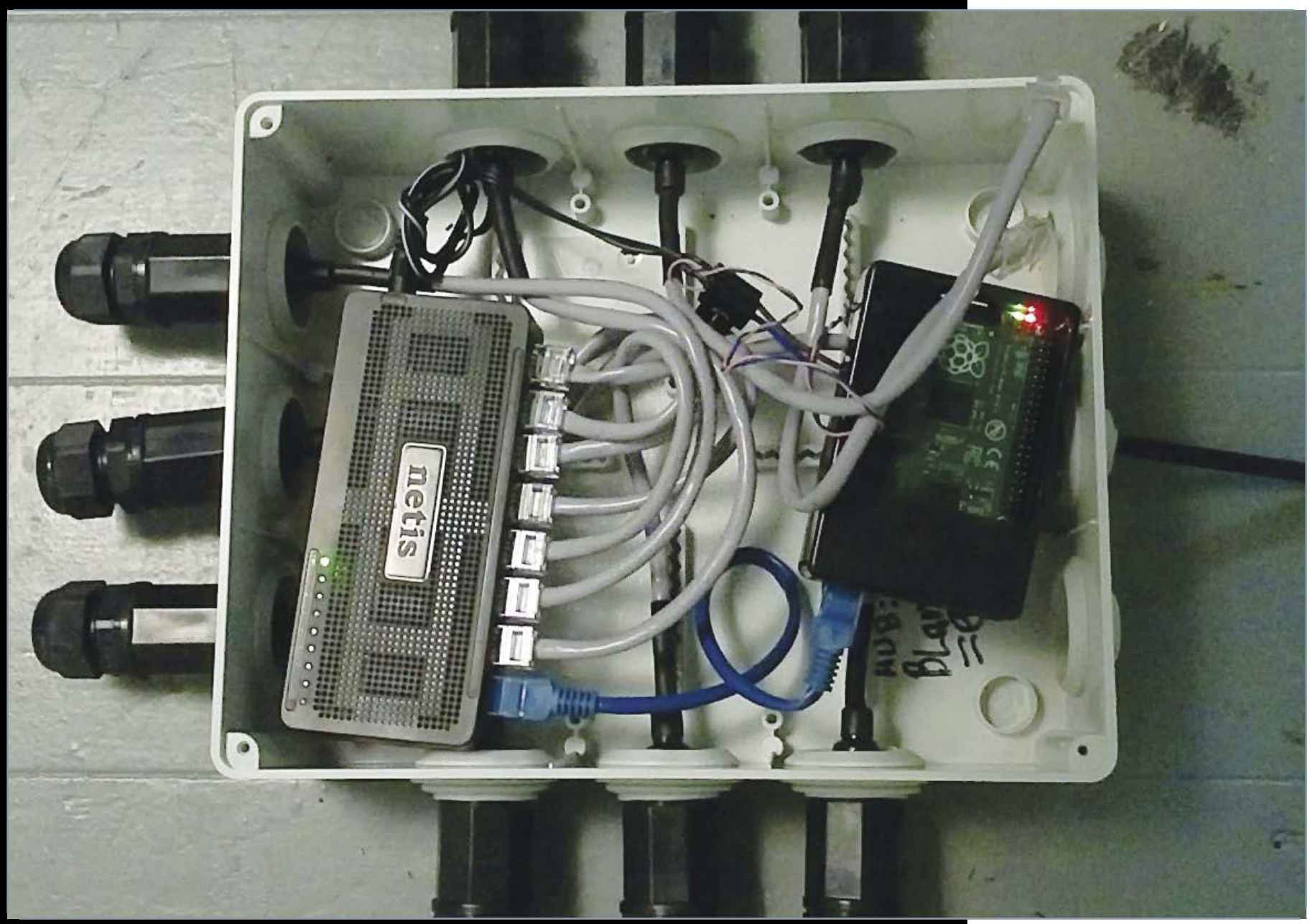
We are very careful about reliability and we have a lot of experience now with making something that is not too expensive, but also that works and sustains the weather and other injuries. Many people will start by driving a WS2812 with Arduinos, which works with one strip, and then to make a screen they will add in more and more. And it will work on the table, but when you move to outdoor installations, the constraints are enormous and Arduino doesn't cut it. So I created a special board, the WizYasep, for driving thousands of LEDs and putting them in parallel to drive even more and make it more reliable.

So you are using a different WizYasep board for each section?

Yann Yes. I used a hierarchical structure: at the top we have a single Raspberry Pi B+, which contains a

Left The custom WizYasep board that Yann designed for the project, which is excellent for driving high-volume LED displays

customised Raspbian. I removed everything to do with X Window, so I saved something like 1 GB, which is even more space for storing the videos. I hardened the operating system so it would be in read-only mode. In these installations, we never know if or when the power supply is removed, so there is the risk of wearing out the SD card if it is storing logs etc. There is also a real-time clock, because we are in a public space and there is the issue of saving energy. When the sun is out, from about 10am to 5pm, the system is turned off. And when the WisYasep boards see that no data is coming, they set everything to black so that it draws less current. The Raspberry Pi is connected with a little 8-port hub, 100



Below The Pi is in a separate enclosure, transmitting the data out to each of the six WizYasep-managed sections of the ceiling

Mbit, so that's okay because one frame is about 50 Kb, and multiplied by 25 frames per second, it's less than 1.5 Mbit per second.

Going back to the display, how are you getting the videos to change and interact with the passengers?
Yann Fred prepares video sequences and then he exports those in a special format, so I can then process them and turn the sequences into files that can be read back. From there, I can modify the program, for example, to apply filters or to speed up or slow down the playback. For now, the system is streamlined, it's smooth, because there is not much processing done on the Pi. It just reads a big block of SD card memory and cuts it into pieces, which are then sent in a special order through the network to the six controller boards, according to a quite simple algorithm that optimises the network congestion. It has a big buffer so it can buffer a lot of data, and then all the screens are updated at the same time by receiving a tiny broadcast packet, so it ensures that the screens are perfectly synchronised.

Like it?

The first project that Fred and Yann worked on together was the Rosace: an electronic persistence of vision mill, similar to a zoetrope or a phenakistoscope. You can check it out here in the first part of the video: http://goo.gl/NvGmcJ

Further reading

Interested in Yann's WizYasep board? There's plenty more information at: goo.gl/eV1dbB

Stream media through your network with OSMC

Set up a Samba server and map a network drive to OSMC to stream media across your home network

Thousands of images, videos and music files live in your pocket and it's likely even more are in your personal cloud or hard drives. While it's great enjoying these on our personal devices, it's gratifying to see them on our big TV screens – but this isn't easy. Do we hook up our laptops? Do we fiddle with a menagerie of wires and resolutions, hoping for a decent image? Will the sound work? All of this is taken care of with a Raspberry Pi 2 and OSMC. OSMC is an OS built to do one thing: put your media up on that big screen. But, how do we get media onto OSMC? Here we need Samba and a networked hard drive.

THE PROJECT ESSENTIALS

OSMC https://osmc.tv

Hard drive

Home network

Samba

Another Linux computer, less than eight years old

01 Install Samba

We're going to assume that you already have a networked OSMC Pi set up already (it's a simple installation), so we'll jump straight into setting up Samba. For this tutorial, we're using Ubuntu 14.04. If you're running a Debian system, all of these instructions should work fine as they are. If not, you may have to search around for the appropriate repos. First, we need to get the Samba package from the repository with apt-get. Open up a terminal and enter:

```
sudo apt-get install samba
```

02 Add a new user to Samba

Once installation has completed, our system now has the ability to share folders and drives with the Samba protocol. Before we can start mapping paths on our home network, we need to create a password. For brevity's sake, we'll use the same user that we're logged in as to manage Samba. Once you enter the following command,

you'll be prompted to enter a password.

```
sudo smbpasswd -a YOUR_USER_NAME
```

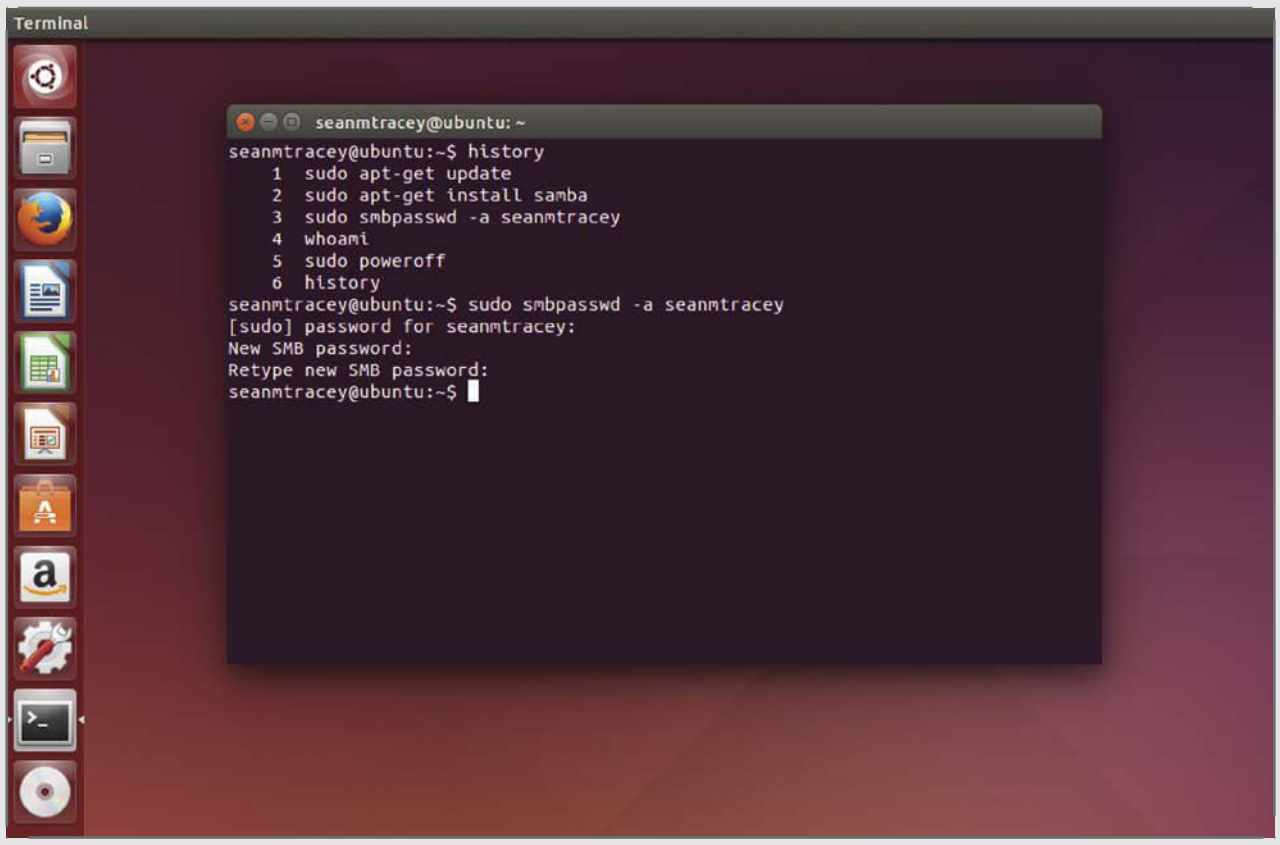
03 Prepare hard drive on the network

In order to mount our volumes for sharing, we need to add some configuration settings to the samba.conf file. *.conf files are quite an important deal, so let's make a copy of ours just in case something goes wrong and we find ourselves tearing out our hair trying to undo the damage.

```
sudo cp /etc/samba/smb.conf ~/Desktop/
```

04 Plug in your drive

If you've had the hard drive you're intending to share plugged-in all this time, don't panic, you've done nothing wrong, but if you haven't then now is the time to hook it up. Open up your file manager window, right click on the drive's icon and click Properties. Take note of the path and volume name.



Left If you run smbpasswd as root, you can also add and remove users from the smbpsswd file

05 Configure Samba

Now that we have a backup of our Samba configuration and we know the path of our media drive, we can set up sharing on our network. Using a command line editor of your choosing (one of our favourites is Vim), open /etc/samba/smb.conf for editing and go to the end of your file:

```
> sudo vi /etc/samba/smb.conf

[DRIVE_NAME]
path = path/to/drive/drive_name
available = yes
valid users = user_you_entered_earlier_for_samba
read only = no
browseable = yes
public = yes
```

Exit and save. Now restart Samba with:

```
> sudo service smbd restart
```

Assuming everything went well, you now have a Samba drive set up for sharing on your home network – so long as it's plugged in.

06 Get the address of the drive

Before we jump over to OSMC, we want to find out the address of our Samba drive. Staying in your terminal, enter ifconfig and you'll get a readout of all of your network interfaces:

```
eth0    Link encap:Ethernet HWaddr
    08:00:27:14:6b:6e
```

Why Samba?

When it comes to network protocols, we're truly spoilt for choice. But why Samba over, say, HTTP or an FTP server to deliver files? Well, in order to get things with HTTP or FTP and so on, you often need to know how to get to them beforehand. That makes complete sense in a server-client world, but Samba has discoverability built-in. Why spend time tweaking URLs and configurations when we can have our devices tell us how to find them?

```
inet addr:192.168.1.5 Bcast:192.168.1.255
  Mask:255.255.255.0
inet6 addr: fe80::a00:27ff:fe14:6b6e/64
  Scope:Link
UP BROADCAST RUNNING MULTICAST MTU:1500
  Metric:1
RX packets:3874 errors:0 dropped:0 overruns:0
  frame:0
TX packets:5394 errors:0 dropped:0 overruns:0
  carrier:0
collisions:0 txqueuelen:1000
RX bytes:737927 (737.9 KB) TX bytes:11086898
  (11.0 MB)
```

You won't get the exact same output, but you should have something like the second line buried somewhere. Look for inet addr: and copy down the IP – this is the address we'll use to access our Samba share over the network. Remember, we're using the Samba protocol, so the address will be (in our case) **smb://192.168.1.5/MEDIADRIVE**. A Samba address is made up of the protocol, IP and name that we gave our Samba share.

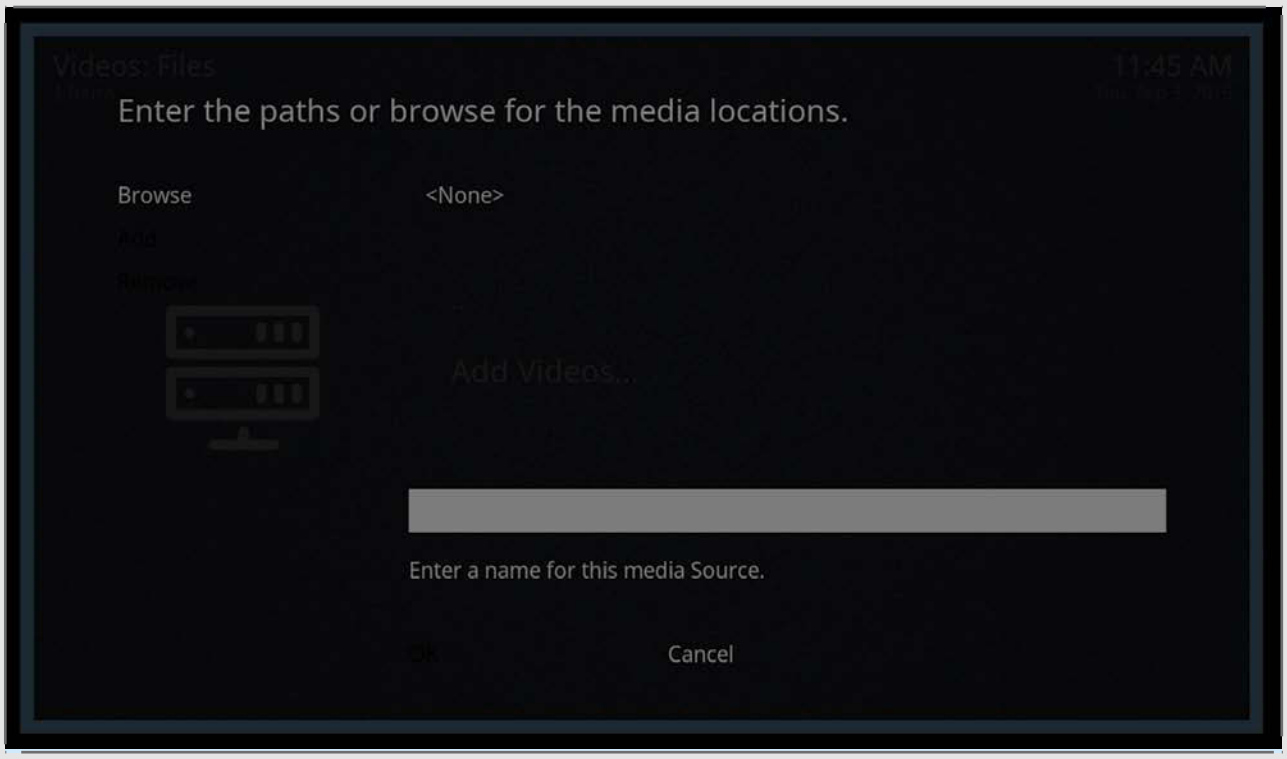
```
smb://   192.168.1.5/ MEDIADRIVE
[PROTOCOL]   [ADDRESSS] [SHARE NAME]
```

07 Add our drive to OSMC

Right, time to leave your Linux box alone – let's do some Linux! Hop on over to your OSMC system. If you're one of the lucky people with the super-cool OSMC remote, you can do everything you need to do with it, but there's quite a bit of typing ahead, so if you have a keyboard lying around then now is the time to plug it in. Power up your

Pi & power

The Pi, even the newer versions, is not renowned for its blistering speed. So how come it can handle streaming and display high-definition content with ease, whereas it might struggle to run a modern version of Chrome and WebGL? Every Raspberry Pi has a hardware H.264 decoder built-in. This specialised chip does one thing really well, really quickly: it plays H.264 video super-fast. That's the ace up the Pi's sleeve that makes it one of the cheapest and best options for home-made media centres like this.



system and go to Videos>Files>Add Videos. You'll be presented with a view like the picture above.

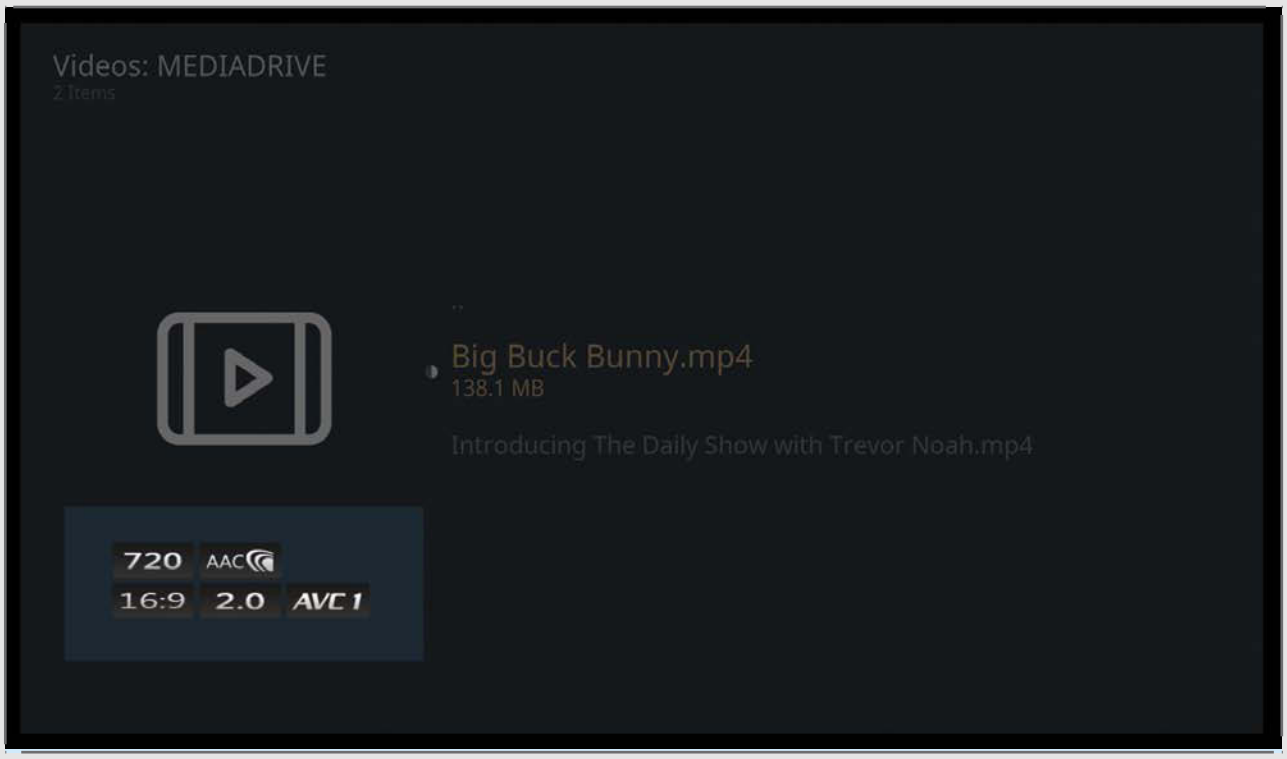
Enter the address we jotted down in the previous step – in our case it would be smb://192.168.1.5/MEDIADRIVE – and hit Done.

08 Appease the password gods

Depending on your setup, you may or may not be asked to enter a username and password. If you do, you'll be prompted as soon as you've entered the path to your media drive – but what is it? That's simple: it's just your ordinary username and password for accessing your system. Once you've entered those, tick the 'Remember this path' box and OSMC will keep a track of the drive and its content as long as it's available on the network.

09 Test the setup

In your videos screen you should now see your shared network drive, which looks just like any other folder. Now that the drive has been mapped, it will reflect any and all changes, structures and files that you apply to your network drive. Select a video to see if it plays. If it does,



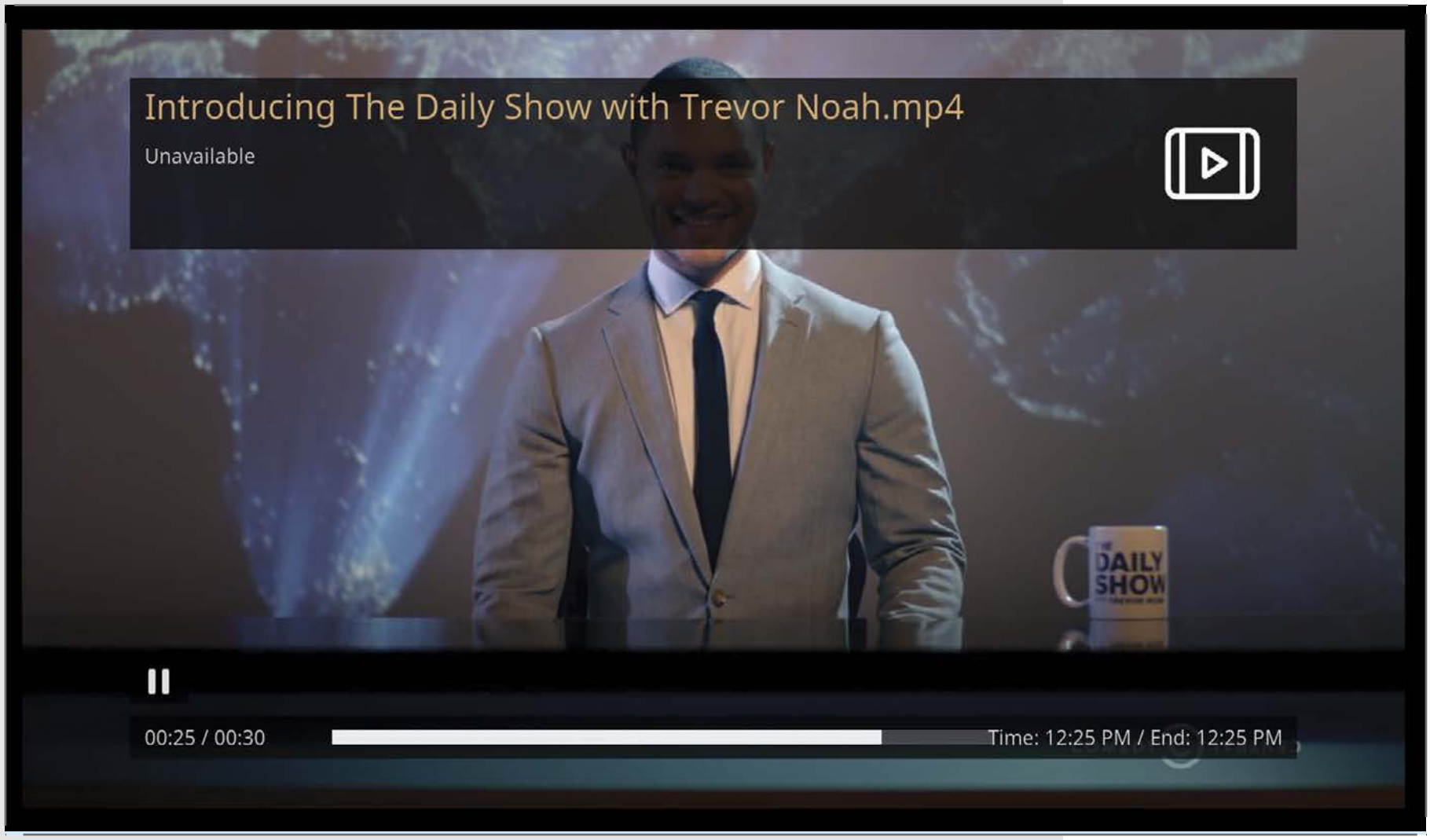
congratulations! You've now got a networked Samba drive to deliver all of your media.

10 Media types and the Raspberry Pi

You may have noticed, just before you tested your video, a bunch of small black boxes with icons and numbers inside them, just to the left of the media selector UI. These display the properties of video files that OSMC is familiar and comfortable with. Don't worry, if you have videos in a slightly unconventional format (Matroska or WebM, for example) OSMC has a ton of goodies built-in to handle things it's unsure of – it might just take a little while for it to figure it out.

11 Add extra media from elsewhere

So, we now have OSMC reading media files from our Samba-serving PC. But what if we want to add more media? Well, OSMC is geared towards watching your media rather than managing it. We could unplug the media drive from our Samba server, plug it into another computer with the media file we want and then plug it

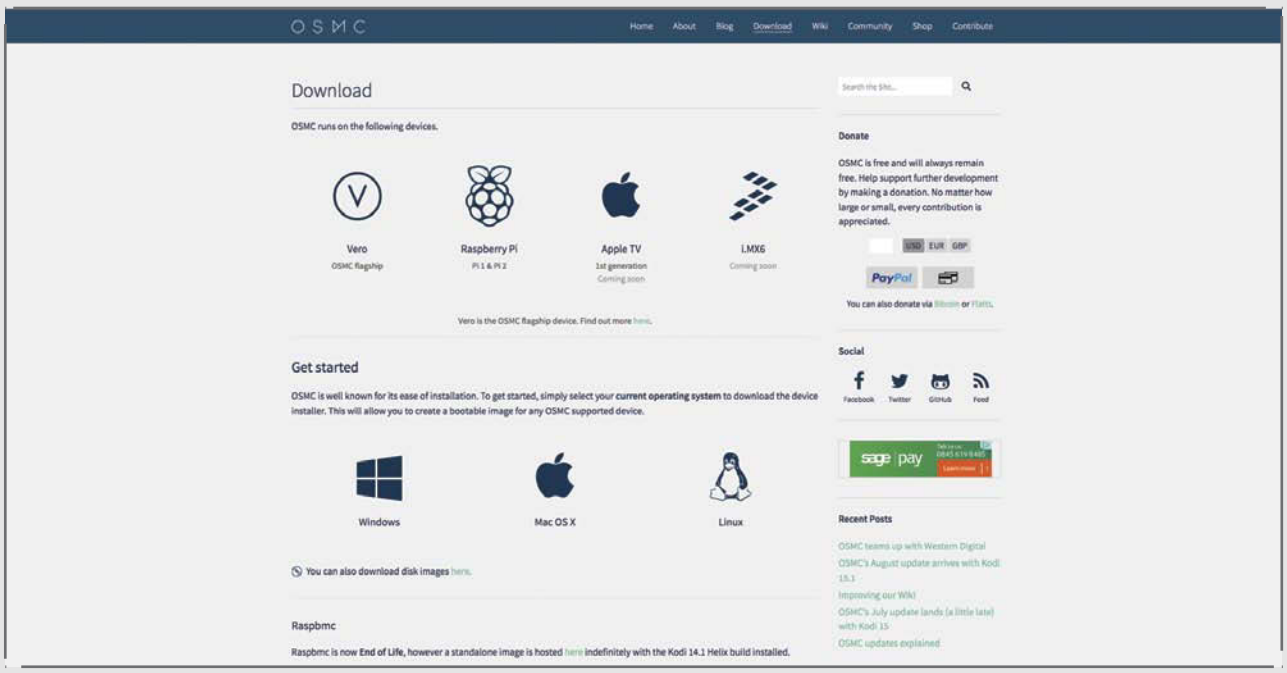


back in to our samba server. Samba can handle plugging drives in and out, but that's cumbersome. Alternatively, we could log in to our Samba server and move files over to it with SCP or FTP, but that's not very user-friendly. What if we could drag media files over the network onto our drive so that we could view them on OSMC?

12 One step forward, seven steps back

Back in Step 5, we set up our MEDIADRIVE Samba share by editing the Samba config file and we have forgotten all about it since. Well, there's more we can do with Samba. As well as making drives readable on our network, we can also make them writable, so let's do that. Open up /etc/samba/smb.conf with your favourite editor and find your drive's Samba configuration (this should be at the very bottom of the file). Right at the end, just after public=yes, add:

Above Don't have an OSMC remote? Try the new Kore remote app from the Kodi devs: http://bit.ly/1JPNs2L



Left To find the Pi SD card images, click the text link just below the three OS logos shown, or go straight to **https://smc.tv/download/images**

```
writable = yes
```

Exit and save.

13 So did that work?

Now we should have a working Samba service running in the background of your computer, but first we need to restart it to make our changes take effect. Before we do that, though, we can check which properties will take effect on restart with the command testparm. Run it as an ordinary user (you shouldn't need to sudo here), press Enter when prompted and look at the output. If you aren't shown any error messages, your config should be fine. So, it is the time to restart Samba again with sudo service smbd restart.

14 Drag, drop, play and enjoy

We now have a network-visible and writable hard drive that OSMC can play files from. You can drag the personal files the you want to watch from one computer on your network onto the Samba drive and have them pop up on OSMC straight away. Let's imagine you use OS X in your day-to-day life (sacrilege!). To add a file

to OSMC, you would only have to open Finder, click Go>Connect to Server, then enter the Samba address for your drive. Next, enter the username and password, just like before, and then drag the file into the window. Now that file is on OSMC and you didn't even have to touch it!

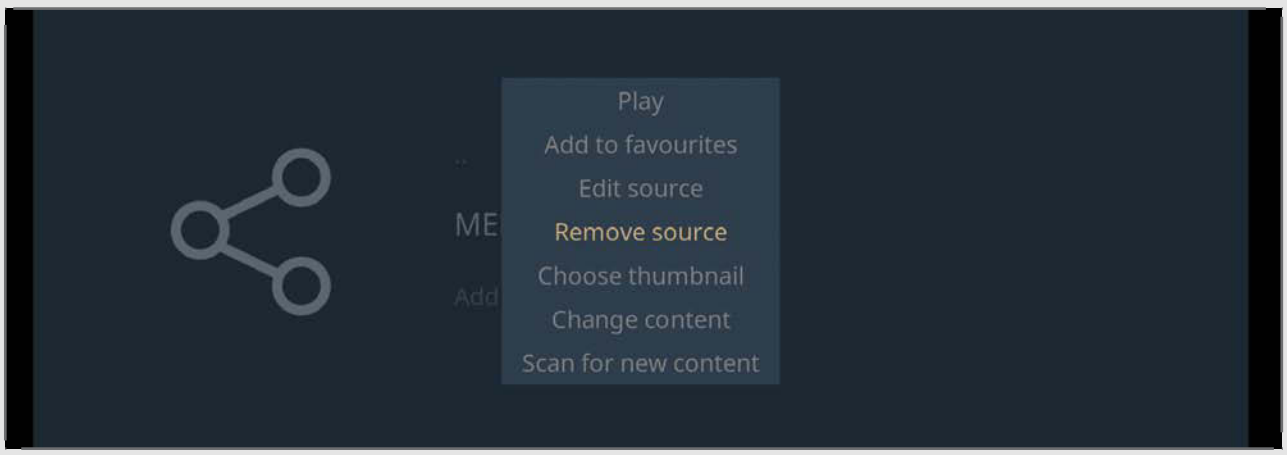
15 More please

If you want to add multiple drives to OSMC, you can repeat Steps 4 through 9. But remember that you don't have to share your entire drive, only a certain folder within your drive – all you have to do is change the path variable to point at a folder. The address will still be the same because it's made up of the IP address and Samba name of your drive. When used properly, this can help you to manage your media over the network.

"Remember that you don't have to share your entire drive, only a certain folder with your drive"

16 When the party is over

If you find yourself wanting to remove your Samba drive as a shared resource in OSMC, it's pretty simple. On OSMC, go to Videos>Files and highlight – but don't click/enter – your Samba drive. Open the context menu (C on your keyboard) and select Remove source. You'll be asked if you're sure and if you want to remove all of the references to the media. If you never intend to use the Samba drive again, you can remove the references.



Harness the power of the 1-Wire bus

Custom sensor circuits require experience in signal processing. The 1-Wire bus simplifies accessing ready-made sensors

When interacting with sensors, one of three buses gets used. SPI is used for high-bandwidth applications, while the slower I2C excels at handling a large amount of slower sensors. Dallas Maxim's proprietary 1-Wire bus has even slower data rates and a longer range than I2C, making it ideal for communicating with tiny devices like weather sensors and thermometers, and it is quite interesting because the 1-Wire bus requires only two wires to be run to the sensor – power is supplied via the data line.

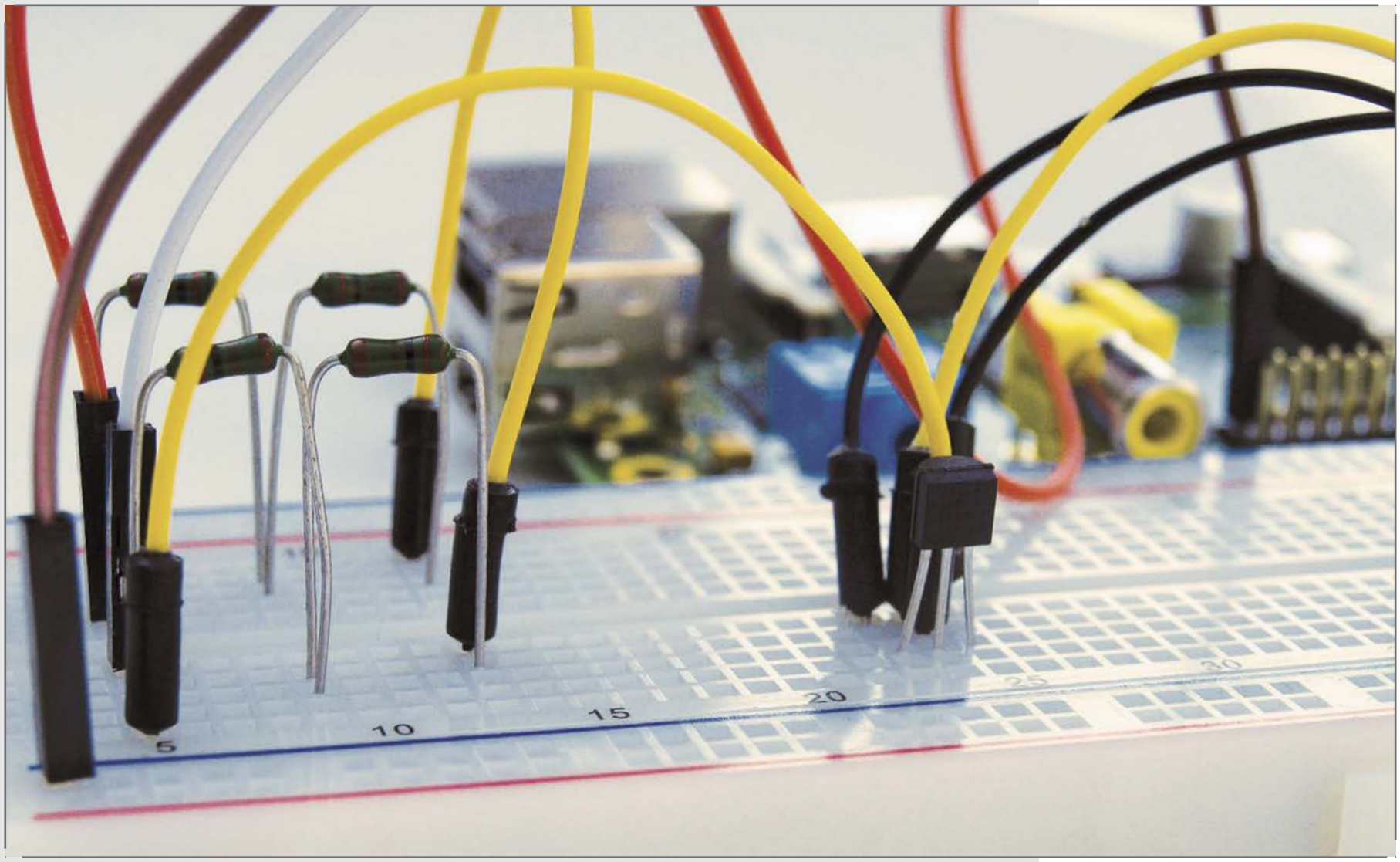
The bus has firmly established itself in two areas of application. First is iButtons, which contain memory and/or a cryptographic ID that can be used for access control – they have been used as keys for residential areas and school campuses, for example, as well as 'smart tickets' for public transportation. In terms of the second main application, Dallas Maxim also peddles a series of inexpensive temperature sensors, which we will take a look at here.

THE PROJECT ESSENTIALS

Dallas Maxim temperature sensors

Breadboard

4.7k resistor



We're going to introduce you to the DS18B20 family. They simplify the gathering of accurate temperature data – bid farewell to AD converters, linearisation and similar hassles.

01 Update your Pi

Dallas Maxim's 1-Wire bus is implemented via a kernel module, which saw significant changes with the introduction of Kernel 3.18.3. Due to this, an update is recommended. Run the commands sudo apt-get update and sudo apt-get upgrade, and check the current kernel version by entering uname -r.

02 Connect the sensors

To measure data, you must connect the sensors to the process computer. Parasitic mode networks are wired up in accordance to the diagram at the top of

the next page. The resistor shown is a pull-up resistor, responsible for making the bus 'float' to 3V3 when it is not pulled down actively.

03 Apply some power

1-Wire is innovative due to its power parasitisation: each sensor has a charged capacitor to provide the energy necessary for measuring. The drive capability of the pull-up resistor is limited, so with larger networks, supply power directly. The kernel driver cannot use the accelerated conversion speed.

04 Configure the kernel

The Raspberry Pi realises a basic form of 1-Wire access via a kernel module. This must be enabled at boot time, a process best accomplished by editing **/boot/config.txt**, as shown below, and then performing a reboot of the process computer. Further guidance on the Pi's config.txt file is available at **http://bit.ly/1K73PqR**.

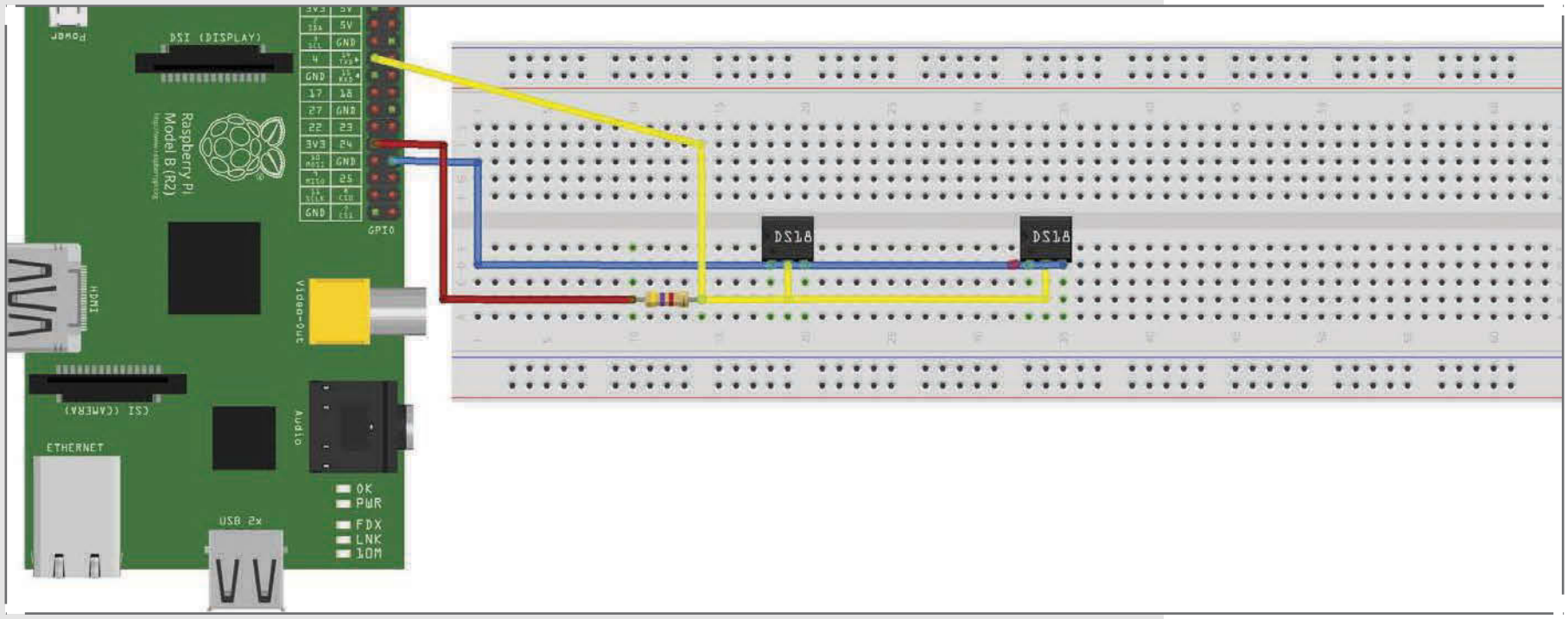
```
dtoverlay=w1-gpio-pullup,pullup=on
dtdebug=on
```

05 Power up

With that in mind, it is now time to power up the process computer. Carefully touch the temperature sensors in order to check if they heat up unduly, as this is a characteristic symptom of wrong connections in the circuit. If it is not the case then your circuit works. You can then look for the sensors in the /sys/bus/w1/devices tree.

Light up

Sensorics can be a funny business. One interesting experiment involves taking two DS18B20 sensors that are placed a few centimetres apart. One of them is then forced to convert data once every second, while the other one gallivants at a more leisurely pace. As time passes by, the active sensor will show a higher ambient temperature. This behaviour is caused by the conversion process, which incidentally generates heat.



Above We're using a 4.7 kΩ pull-up resistor in this circuit

06 Read with C

W1-therm maps the individual devices into the file space. This means that the data contained in them can be read like any other text file. We will deploy a small program that starts out by traversing the filesystem in order to find eligible devices.

```
#include <stdio.h>
#include <dirent.h>
#include <string.h>

void main()
{
  DIR *dir;
  dir=opendir("/sys/bus/w1/devices");
  struct dirent* mydir;
  while((mydir=readdir(dir)))
  {
      if(mydir->d_type==DT_LNK && strstr(mydir-
>d_name, "28-"))
    {
      printf(mydir->d_name);
```

```
        printf("\n");
    }
  }
  closedir(dir);
}
```

07 Read with C, part 2

In the next step, the inner loop must be expanded. It will iterate over the various devices, reading their data using the normal C APIs. Data is then printed to the command line in a slightly formatted fashion.

```
if(mydir->d_type==DT_LNK  &&  strstr(mydir->d_
    name,
"28-"))
  {
      char myField[1024];
      sprintf(myField, "/sys/bus/w1/devices/%s/
  w1_slave", mydir->d_name);
      FILE *myFile=fopen(myField,"r");
          if(myFile!=NULL)
          {
              fscanf(myFile, "%*x %*x %*x %*x
  %*x
  %*x %*x %*x %*x : crc=%*x %*s");
              double myTemp;
              fscanf(myFile, "%*x %*x %*x %*x
  %*x
  %*x %*x %*x %*x t=%lf", &myTemp);
              myTemp/=1000;
              printf("%lf \n",myTemp);
          }
      }
```

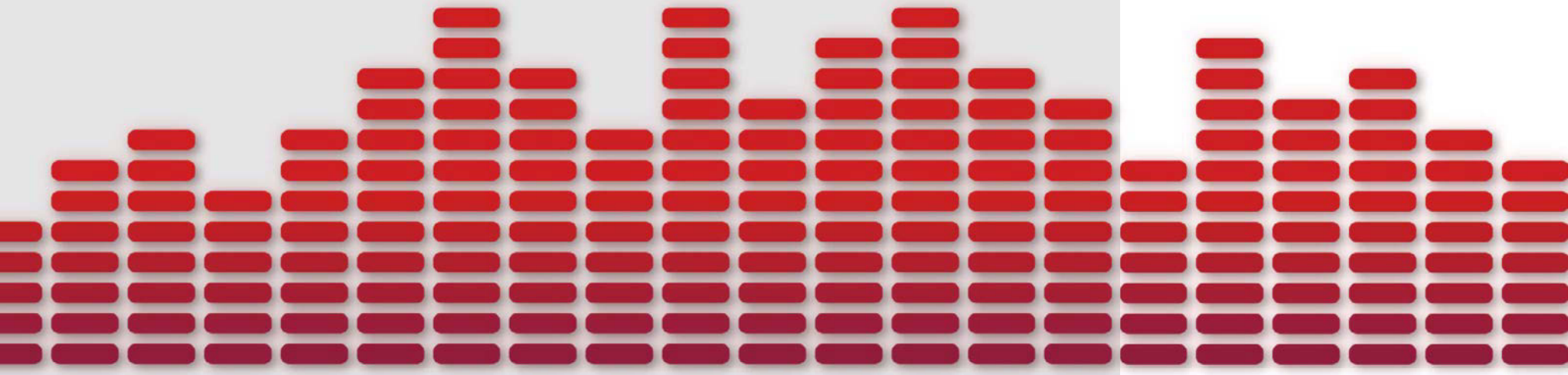
08 Compile and run

Developers who are used to working with Python – the scripting language can, of course, also be used to access the file – might find the deployment process a little odd. GCC compiles our code into an executable file, which can then be run like any other application.

```
pi@rpilab ~ $ gcc rpiTherm.c
pi@rpilab ~ $ ./a.out
27.375000
27.500000
```

09 Go pro!

Accessing professional 1-Wire devices – think memory chips, iButtons and cryptographic coprocessors – is handled via the OWFS library. Unfortunately, this does not support the w1-gpio library. Using it will require the deployment of expansion hardware: the I2C-based DS2482 is the most common external controller.



Send an SMS from your Raspberry Pi

Create a program that combines Twilio and simple Python code to enable you to send an SMS (text message) from your Pi to a mobile phone

Text messaging, or SMS (Short Message Service), has become a staple of everyday communication. What began life as a 40 pence message service is now offered by most tariff providers as an unlimited service. Twilio, a cloud communications company, enables you to send SMS messages for free from your Raspberry Pi to a mobile phone using just six lines of code.

THE PROJECT ESSENTIALS

Raspberry Pi

Twilio account

01 Set up your Twilio account

The first step of this project is to register for a Twilio account and Twilio number. This is free and will enable you to send an SMS to a registered, verified phone. Once signed up, you will receive a verification code via SMS to the registered phone. When prompted, enter this onto the Twilio site to authenticate your account and phone. Go to twilio.com/try-twilio and create your account.

02 Register and verify mobile numbers

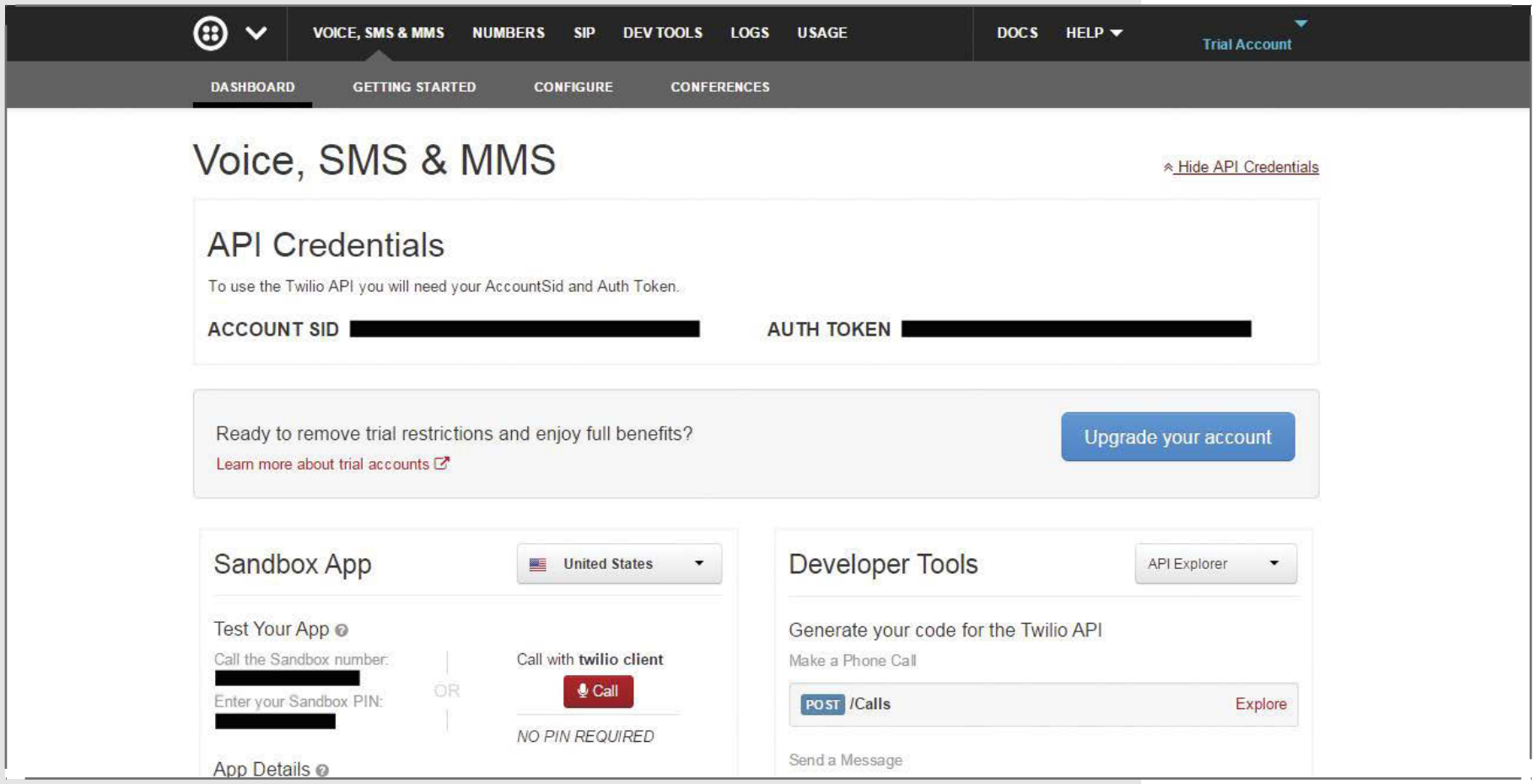
Your Twilio account is a trial account (unless you pay the upgrade fee), which means you can only send and receive communications from a validated phone number. Enter the phone number of the mobile that you want to verify, ensuring that you select the correct country code. Twilio will text you a verification code. Enter this code into the website form and press submit.

03 The dashboard

Once registered and logged in, visit the dashboard page, which will display your AccountSid and your Auth Token. These are both required to use the Twilio REST. Keep these secure and private, but be sure to make a note of them as you will need them for your Python program later.

REST

REST stands for Representational State Transfer. (It is sometimes spelt "ReST".) It relies on a stateless, client-server, cacheable communications protocol – and in virtually all cases, the HTTP protocol is used. REST is an architecture style for designing networked applications.



Above You will be able to find your AccountSid and your Auth Token on the Twilio dashboard

04 Install the software

Boot up your Raspberry Pi and connect it to the Internet. Before you install the Twilio software, it is worth updating and upgrading your Pi. In the LX Terminal, type sudo apt-get update, then sudo apt-get upgrade. Once complete, type sudo easy_install twilio or sudo pip install twilio to install the software. (If you need to install pip, type sudo apt-get install python-pip python-dev, press Enter, then type sudo pip install -U pip.)

05 Twilio authentication

Now you are ready to create the SMS program that will send the text message to your mobile phone. Open your Python editor and import the Twilio REST libraries (line one, below). Next, add your AccountSid and Auth Token, replacing the X with yours, as you will find on your dashboard:

```
from twilio.rest import TwilioRestClient
account_sid = "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
    # Enter Yours
auth_token = "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
    # Enter Yours
client = TwilioRestClient(account_sid, auth_
    token)
```

06 Create your message

You will probably want to be able to change your text messages rather than send the same one. Create a new variable in your program called message. This will prompt you to enter the phrase that you want to send to the mobile phone. When the program runs, this is the message that will be sent:

```
message = raw_input("Please enter your
    message")
```

07 Add your numbers

To send the message, you need to add the code line below and your two phone numbers. The first number is your mobile phone number, which is registered and validated with Twilio (Step 2). The second number is your Twilio account number, which can be retrieved from your dashboard page under 'Call the Sandbox number'. Change the Sandbox number to your country location and remember to add the international country code.

```
message = client.messages.
    create(to="+44YOURMOBNUMBER",
    from_="+44YOURTWILIONUMBER", body=message)
```

"Twilio provides a wide range of API codes and reference documents to create other communication programs"

08 Send the message

Now send your message. The code below is not required, but useful to indicate your message has been sent. Add the lines and save your program. Ensure your Raspberry Pi is connected to the Internet and that your mobile is on, then run your program. You have just texted from your Raspberry Pi!

```
print message.sid
print “Your message is being sent”
print “Check your phone!”
```

09 Other API and codes

Twilio provides a wide range of API codes and reference documents to create other communication programs, such as making phone calls, recording a call, and retrieving data including caller IDs and call duration. The API also complements a wide range of languages, including Ruby, PHP, Java and Node.js (https://www.twilio.com/api).

Access Twitter with Python

Twitter can be a great communication channel to enable your Raspberry Pi project to talk to the world

We've looked at how to turn your Raspberry Pi into an RSS feed reader before and how to enable your Raspberry Pi to post an RSS feed for others to read. In this issue, we will continue along the same lines and look at how to add the ability to have your Raspberry Pi post tweets for the rest of the world to follow what is happening.

The first step is to find some way to talk to Twitter. Twitter provides a full API that defines how you can send and read tweets. We could work with raw sockets; manually format and send messages, then process the returned results. However, this completely defeats the whole point of Python and the premise of code reuse. Instead, we will use the module python-twitter. There are several other options available too, in case you wish to look for a module that uses a different code style that better matches your own. They should all have the same functionality, so feel free to explore and experiment. For these examples, you can install the module with pip install python-twitter. This works fine for Python 2.x, but Python 3 support is something that is in progress, so you may need to pick a different module if you want to use Python 3. If you need the latest features then you can always grab the source code

Above Consider setting up a fresh Twitter account specifically for your project

from GitHub and build it.

Once you have the module installed, you can import it within your code with import twitter. The class twitter.API provides the main interface to enable you to interact with the Twitter API. When you create a new instance of the class, you need to provide your credentials in order to authenticate to Twitter and be allowed to work with your account. Twitter now uses OAuth as the main way to handle authentication, and this means that you need to generate an access token that your code can use to identify itself with. Twitter has an entire section within its developer documentation that covers the steps required to get your token, located at goo.gl/99qAfa. The first step is that you need to register your code as an application that is authorised to talk with Twitter. You handle this by going to apps.twitter.com and creating a new app. This gives you the consumer key and consumer secret that identifies your code as being permitted to connect to Twitter. The second step is to create an authentication token for your account that will tell Twitter that this app is authorised to connect to your account, read it and post to it. You should end up with two separate keys and secrets that you can now use to create an instance of the

API class. The code looks like:

```
import twitter
  api1 = twitter.API(consumer_        key='consumer_
    key',          consumer_secret='consumer_
    secret', access_token_
    key='access_token',
    access_token_secret=
    'access_token_secret')
```

... where the relevant keys and secrets are the ones that you have generated above. You can check to see whether these credentials worked by calling the function api1.VerifyCredentials(). This will either give you an error or give you a breakdown of the account information.

Once you have an authenticated connection to Twitter, your Raspberry Pi project can start to talk to the world at large. The most basic method is PostUpdate(). With this function, you can send in a status update message, with a maximum of 140 characters. If your message is longer than that, you can use the function PostUpdates(), which will post multiple status updates until the entire message is posted. If you want to, you can include other details with your message. This includes options like the current longitude and latitude, or a place ID, to identify where a particular update came from. This might be handy on a mobile monitoring project of some kind where you want to know where a particular update came from. You can also post an update as a response to some other message, by including the in_reply_to_status_id= option in the function call. If you don't want to use public status updates for your messages, you can use the function PostDirectMessage() to send a text message to a particular account. You can use

Why Python?

It's the official language of the Raspberry Pi. Read the docs at https://www.python.org/doc

either the user_id or the screen_name to identify the recipient of your direct message. While this is a bit more private than a public update, it is by no means secure, so don't send any compromising information using this method. If you want to, you can also send multimedia files as Twitter updates. The function PostMedia() can take either a local file name or an HTTP URL to do a multimedia tweet. You can send image files like JPEG, GIF or PNG. You can also include an associated text message for the tweet, as well as location information if you want to geolocate the tweet and image. Once you finish any of these posting functions, you get a status object returned that gives you details of your status. There are two dozen elements giving you items, such as whether it has been favourited, when it was created and the hashtags associated.

The Twitter API also enables you to read updates and direct messages from other users. This means that you can use this same communication channel to be able to send commands to your Raspberry Pi project. The function GetDirectMessages() can get up to the last 200 direct messages sent to your Raspberry Pi's Twitter account. Luckily, you don't have to do this every time. You can use the option since_id=X to tell Twitter to only send those direct messages that are newer than message ID X. If you are only interested in getting the last Y messages, you can instead use the option count=Y. This function returns a sequence of DirectMessage object instances. This structure contains the message ID, the creation date, recipient and sender information, as well as the actual message's text. This way, you can send commands within the text message so that your project can do different tasks after it has been deployed.

With the examples we have here, you should be able to have your project tweet what is going on out to the world.

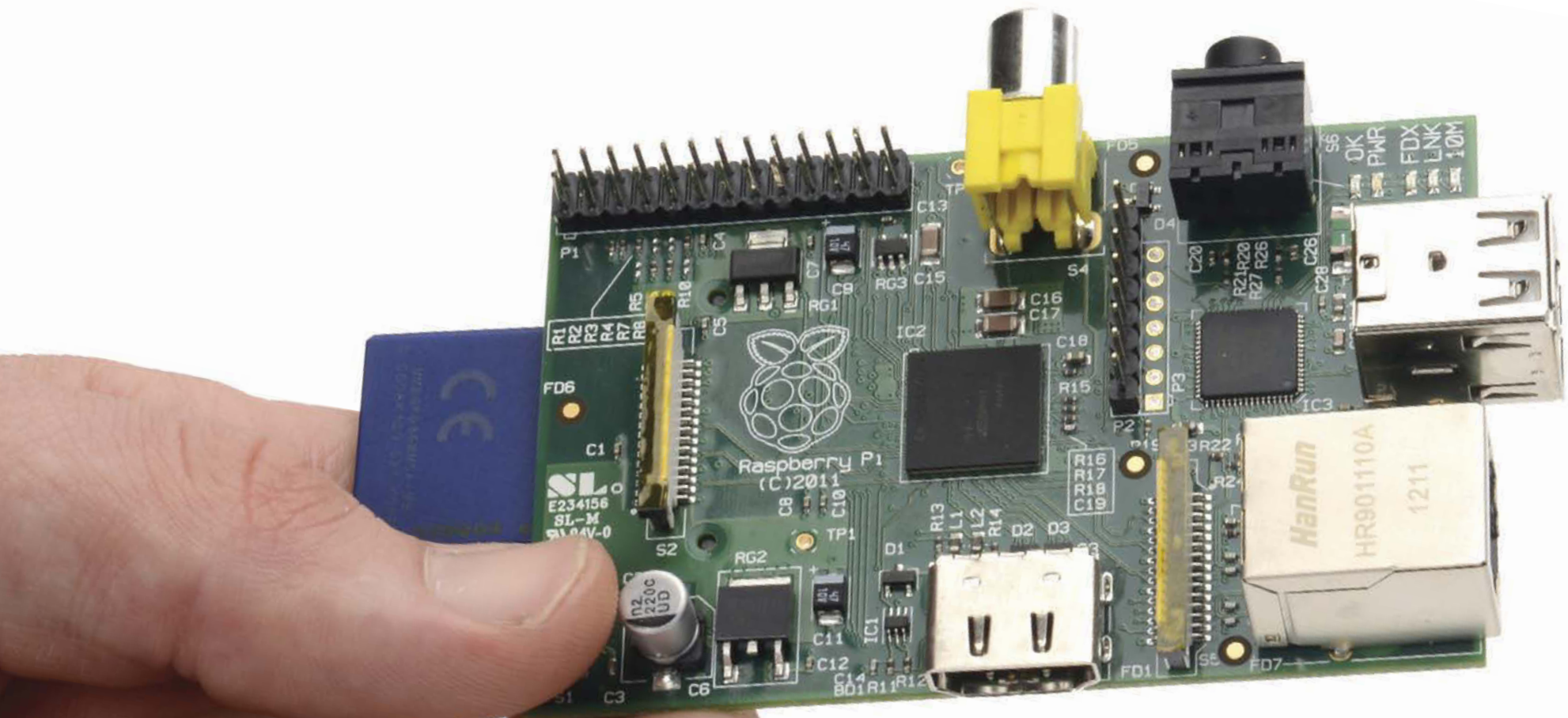
The Code ACCESS TWITTER WITH PYTHON

```
import twitter

# The first step is to connect to Twitter
api1 = twitter.Api(consumer_key='consumer_key',
          consumer_secret='consumer_secret', access_token_
          key='access_token', access_token_secret='access_
          token_secret')

# You can post status updates
update_status = api1.PostUpdate('I am now connected!')

# You can geotag updates with lat/long
update_status = api1.PostUpdate('I am here: ',
latitude=45.949874, longitude=-66.642347)

# Posting images taken by your RPi
update_status = api1.PostMedia('This is my picture',
'image1.png')

# Get the last 10 mentions and retweets
mentions = api1.GetMentions(count=10)
retweets = api1.GetRetweetsOfMe(count=10)

# Finding out how many followers you have
followers = api1.GetFollowers()
follower_cnt = len(followers)
```

Talking Pi

Join the conversation at...

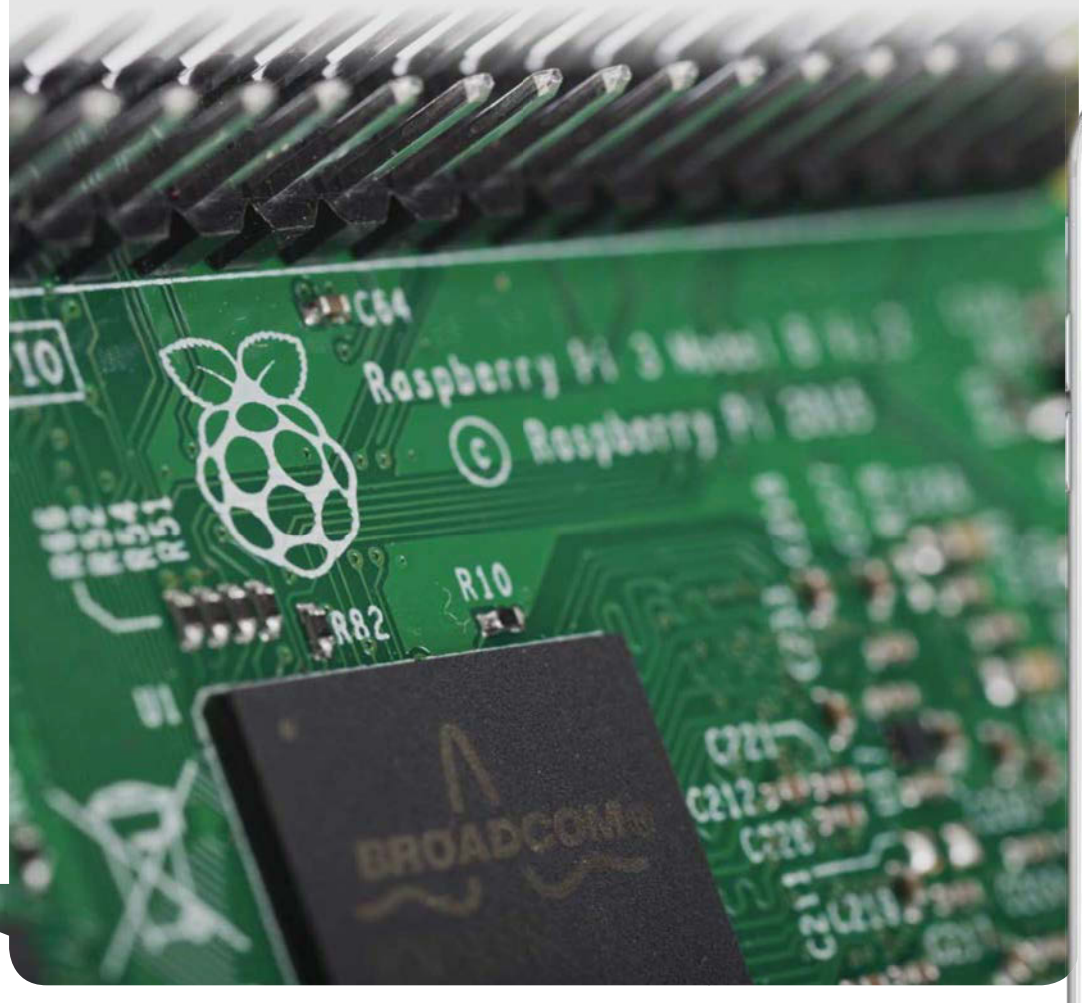
@linuxusermag | Linux User & Developer | @ RasPi@imagine-publishing.co.uk



The Raspberry Pi encourages a lot of hands-on work and this means it doesn't quite work like – or as easily as – your laser focus-tested smartphone interface, or even a normal computer. So we're answering your burning Raspberry Pi questions each issue – get in touch with us on Twitter, Facebook or by email.

Can I run other operating systems on the Pi apart from Raspbian or is that the only one that's available?
Callum via email

Hi Callum. No, you're not limited to Raspbian by any means! It's the official one for the Raspberry Pi, true, but it's not the only one that you can use. Head over to the Raspberry Pi official website at **https://www.raspberrypi.org/downloads/** and you'll see a range of operating systems for you to enjoy, including Ubuntu Mate and even RiscOS (if you're feeling particularly old-school). And these aren't the only ones available either. In issue 167 of our sister magazine **Linux User & Developer** we ran a version of Kali Linux, the security-testing distro, on the Raspberry Pi. You can get it from **http://docs.kali.org/kali-on-arm/install-kali-linux-arm-raspberry-pi**.



Keep up with the latest Raspberry Pi news by following @LinuxUserMag on Twitter. Search for the hashtag #RasPiMag

JUST A SCORE
WHAT'S YOUR JUST A SCORE?

Have you heard of Just A Score? It's a new, completely free app that gives you all the latest review scores. You can score anything in the world, like and share scores, follow scorers for your favourite topics and much more. And it's really good fun!

Why doesn't the Raspberry Pi support VGA at all?

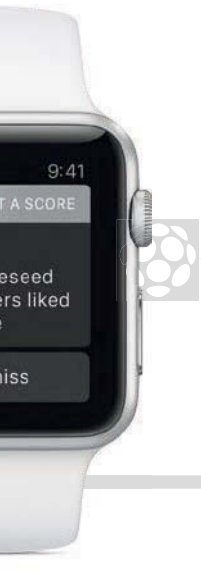
Derek via email

VGA (Video Graphics Array) is a technology that's very much at the end of its life (it came out in 1987, after all), so the Pi supports HDMI as standard. If you've got an old screen you want to use though, it is possible to get the Raspberry Pi to output to a VGA device. You'll need a HDMI to VGA adaptor (you can pick them up for between £5 and £20 if you shop around online). The recommended model is just £6. https://www.pi-supply.com/product/gert-vga-666-hardware-vga-raspberry-pi/

Can I power my Raspberry Pi using a USB hub or not?

Kelly via email

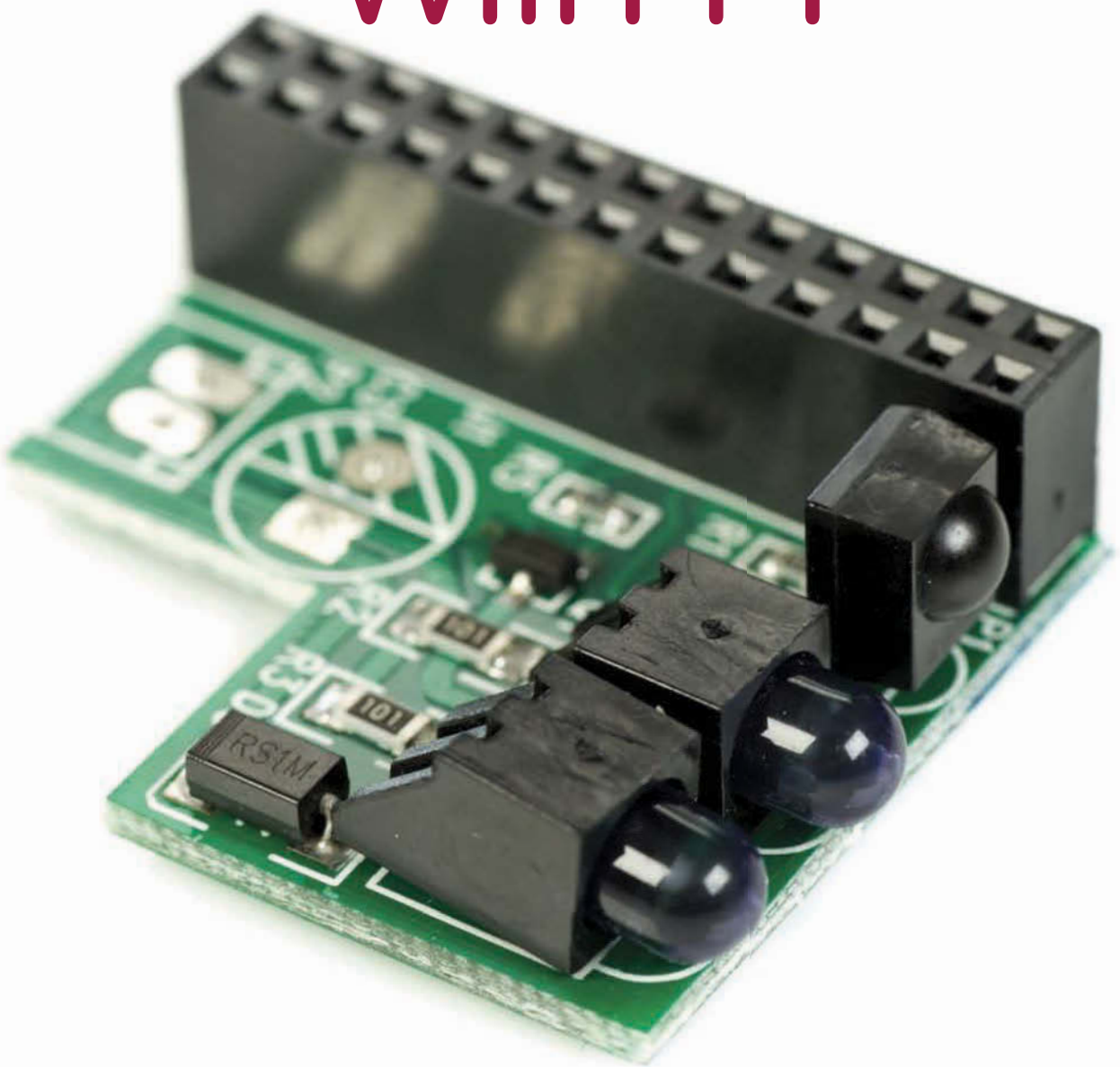
You can, but whether you should is another matter. Some USB hubs will power a Raspberry Pi with absolutely no trouble. Some won't. And some will cause an issue called 'backfeeding'. They'll power the Pi, sure enough, but they'll bypass its power surge protection, so if the USB hub has a power surge then it's bye bye Miss USB-hub-powered Pi. And of course, there are so many USB hubs in the world that it would be an impossible task for us (or indeed anyone) to say with certainty that a particular make and model are ideal, or should never be used. So, inconvenient as it may be, we recommend that you don't.



Next issue

Get inspired **Expert advice** **Easy-to-follow guides**

Hack your TV with Pi



Get this issue's source code at:
www.linuxuser.co.uk/raspicode